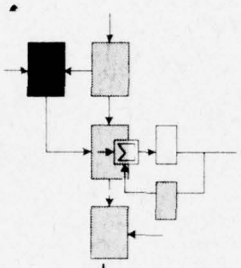
March, 1977

ESL-R-756
Contract ONR/N00014-75-C-1183



COMPARATIVE ANALYSIS OF ROUTING ALGORITHMS FOR COMPUTER NETWORKS

Joe Edmund Defenderfer

Electronic Systems Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139

Department of Electrical Engineering and Computer Science

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

(6) COMPARATIVE ANALYSIS OF ROUTING ALGORITHMS FOR COMPUTER NETWORKS.

5. TYPE OF REPORT & PERIOD COVERED

(9) Technical Report.

6. PERFORMING ORG. REPORT NUMBER

(14) ESL-R-756

7. AUTHOR(s)

(10) Joe Edmund Defenderfer

8. CONTRACT OR GRANT NUMBER(s)

ARPA Order No. 3045/5-7-75
ONR/N00014-75-C-1183

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Massachusetts Institute of Technology
Electronic Systems Laboratory
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

Program Code No. 5T10
ONR Identifying No. 049-383

11. CONTROLLING OFFICE NAME AND ADDRESS

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

(11)

12. REPORT DATE

June 1977

13. NUMBER OF PAGES

155

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

Office of Naval Research
Information Systems Program
Code 437
Arlington, Virginia 22217

(12) 156 p.

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(13) N00014-75-C-1183,

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

√√ ARPA Order - 3045

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network
Computer
Communication
Routing
Load-Sharing
Algorithm

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The routing problem in a computer-communication network is modeled as a multicommodity flow problem. Two generalizations of the EF algorithm of Cantor and Gerla for solving the nonlinear multicommodity flow problem are presented. These generalizations are of interest because they retain identity of the components of the solution in an implicit and compact manner.

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

127200

4B

20. A mini-max routing problem is defined as a linear program, and a methodology for its solution as both a linear and nonlinear program is presented. The two approaches are compared with each other, and a lower bound on the solution is developed for the nonlinear approach.

Two shortest route algorithms which are very efficient for classes of topologies likely to be considered in routing problems are introduced. The shortest route problem is a subproblem of all of the routing algorithms mentioned above.

The computer load sharing problem is formulated as a multicommodity flow problem. A algorithm is presented to solve the load sharing problem which exploits the special structure of the problem.

COMPARATIVE ANALYSIS OF ROUTING ALGORITHMS FOR COMPUTER NETWORKS

by

JOE EDMUND DEFENDERFER
B. E. E., Georgia Institute of Technology
(1973)
M.S., The Ohio State University
(1974)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
March 1977

Signature of Author: Joe Edmund Defenderfer
Department of Electrical Engineering and
Computer Science, March 3, 1977

Certified by: .....
Thesis Supervisor

Accepted by: .....
Chairman, Departmental Committee on Graduate Students

COMPARATIVE ANALYSIS OF ROUTING ALGORITHMS FOR COMPUTER NETWORKS

by

JOE EDMUND DEFENDERFER

Submitted to the Department of Electrical Engineering and Computer Science on March 3, 1977, in partial fulfillment of the requirements for the Degree of Doctor of Science.

ABSTRACT

The routing problem in a computer-communication network is modeled as a multicommodity flow problem. Two generalizations of the EF algorithm of Cantor and Gerla for solving the nonlinear multicommodity flow problem are presented. These generalizations are of interest because they retain the identity of the components of the solution in an implicit and compact manner.

A mini-max routing problem is defined as a linear program, and a methodology for its solution as both a linear and nonlinear program is presented. The two approaches are compared with each other, and a lower bound on the solution is developed for the nonlinear approach.

Two shortest route algorithms which are very efficient for classes of topologies likely to be considered in routing problems are introduced. The shortest route problem is a subproblem of all of the routing algorithms mentioned above.

The computer load sharing problem is formulated as a multicommodity flow problem. An algorithm is presented to solve the load sharing problem which exploits the special structure of the problem.

Thesis Supervisor: John M. Wozencraft

Title: Professor of Electrical Engineering

ACKNOWLEDGEMENTS

It has been an honor and a pleasure to be acquainted with and have my thesis supervised by Professor John M. Wozencraft. I am very appreciative of the freedom which he allowed in the pursuit of the goals of the research which he directed, of the advice which he gave on the content and presentation of this thesis, and of the many discussions that we had during my graduate research at M.I.T.

I am thankful to the readers of this thesis, Professor Robert G. Gallager and Dr. Stanley B. Gershwin, for the helpful discussions and comments they have provided from before the time at which this thesis was proposed until the present.

I am grateful to my fellow graduate students, Mr. Eberhard Wunderlich and Mr. Pierre Humblet, with whom I shared many brain-storming sessions, the evidence of which is distributed throughout this thesis. Much of Chapter V was done in collaboration with Mr. Wunderlich whose M.S. thesis was on the topic of load sharing in a system of computers. The example of Fig. 5.2.3 is owed entirely to Mr. Humblet.

I thank Ms. Delphine Radcliffe for her skillful typing of this thesis. Her swiftness and accuracy made the completion of this manuscript far less tedious than expected.

I wish to thank my parents for their encouragement and financial support throughout my student career. I also wish to acknowledge support by a Sloan Research Traineeship and by the Advanced Research Projects Agency under contract ONR/N00014-75-C-1183.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

CHAPTER I

INTRODUCTION

The purpose of this thesis is to investigate and compare some of the algorithms which may be used to find routing strategies in a computer-communication network. Under a variety of assumptions first proposed by Kleinrock [KLEI64] which allow analytic modeling of packet switched computer-communication networks, the static routing problem can be formulated as a nonlinear multicommodity flow problem. By the mechanisms used to solve such problems, one can find routing strategies which minimize an objective function such as expected delay through the network of a randomly selected packet. The form of a routing strategy is a set of parameters which specify the fraction of packets arriving at each node which is to be sent through each outgoing channel from the node given the packet destination.

Chapter II investigates the minimum delay problem in packet switched computer-communication networks. Emphasis is on the EF (Extremal Flow) algorithm of Cantor and Gerla [CANT74] and generalizations of that algorithm. The generalizations of the EF algorithm are interesting because they convey the detail of the solution, i.e. the actual makeup of the solution from which a routing strategy is derived, in an explicit and compact way. The chapter discusses the complexities and memory requirements of the various algorithms in a comparative manner.

Chapter III introduces two shortest route algorithms which are efficient for certain interesting classes of networks. The shortest route problem is an important subproblem of many network problems, e.g. the EF algorithm.

Chapter IV discusses a "throughput" problem in capacitated networks. The problem can be solved exactly via a linear programming approach; however, the linear program can require a great deal of computation time on some relatively simple networks. An alternative is to approximate the solution through a sequence of solutions to nonlinear multicommodity flow problems. The tradeoffs between these two approaches are discussed.

Chapter V is concerned with the computer load sharing problem. The load sharing problem arises when a set of computers which are serving local users in a batch processing mode are interconnected via communication channels to share their resources for better overall usage of the facilities. The algorithm of that chapter provides the routing strategy which minimizes the expected time for passage through such a system for a randomly selected job. The algorithm is a special case of those of Chapter II; however, various aspects of the problem are exploited to increase the efficiency of the algorithm.

The remainder of this introductory chapter is primarily concerned with the nature of multicommodity flows and the solution of linear multicommodity flow problems in the absence of capacity constraints. These concepts enhance the understanding of the decomposition methods for solving nonlinear multicommodity flow problems

and for solving linear multicommodity flow problems with capacity constraints. The final section of this chapter deals with the format of routing policies, since an optimal routing policy is the goal of all routing algorithms.

1.1 Multicommodity Flows

The multicommodity flow problem will be stated in this section in the terms which will be most useful in later chapters for presenting the body of this report. The networks over which the problems are defined are assumed to be directed and connected. Define:

r_{ij} = value of flow to be sent from node i to node j (one sets $r_{ii} = 0$)

f_i = flow on arc i

N = number of nodes in the network

NA = number of arcs in the network

C_i = capacity of arc i

In the above, arcs are denoted by a single index, but sometimes it will be convenient to use the notation arc(i, j) for the directed arc from node i to node j. If the origin and destination of arc k are OR(k) = i and DS(k) = j, respectively, then arc k corresponds to arc(i, j). Arcs which have no capacity are assumed not to exist, so that NA represents the number of arcs with capacity greater than zero.

There are three basic types of constraints in dealing with network flows: nonnegativity of flows, conservation of flows, and upper bounds on flows. The remainder of this chapter deals with only the

first two types of constraints, and it is assumed that if the capacity of an arc is greater than zero then the flow through the corresponding arc is allowed to be unbounded. In later chapters, the capacity constraints will be included implicitly or explicitly as required.

Writing the conservation equations and nonnegativity constraints for a general directed network is a difficult problem unless one employs commodity distinction in the constraints (as will be done later in this section). In fact, a systematic method for writing the required equations probably has not been documented. Some of the conservation constraints may be written in this manner: (i) choose any set of arcs and denote this set A, (ii) find the set B of origin-destination pairs that no longer have a path between them if the set A is removed from the network, and (iii) write the corresponding conservation equation:

$$\sum_{i \in A} f_i \geq \sum_{(j,k) \in B} r_{jk}$$

One makes the assumptions that all conservation equations may be written (and that there are P of them), that they are linear constraints, and that they may be written along with the nonnegativity constraints as

$$Mf \geq b, \quad f \geq 0 \qquad (1.1.1)$$

where $f = [f_1, f_2, \cdots, f_{NA}]'$, $b = [b_1, b_2, \cdots, b_P]'$, and M has dimension P x NA. Vector inequalities are interpreted in a fashion similar to the following throughout this thesis: $f \geq 0$ means $f_i \geq 0$ for every

$i \in \{1, 2, \cdots, NA\}$. The set of all flows f that satisfy 1.1.1 is referred to as the set F.

The reader should not be disturbed by this rather imprecise method of definition of F. The development of F in this manner has had these objectives: (i) to point out that there is no existing systematic method of generating the constraints for F in a format which does not introduce variables other than those representing total flows on the arcs, in the hope that someone will be inspired to solve that problem, and (ii) to provide convenient notation. It is well established [ASSA75] that the set F is a convex polyhedron which implies the existence of a constraint equation such as 1.1.1.

Any questions as to the definition of F are answered by the more precise definitions of F provided by the following two constraint formats. They allow a compact representation of the constraints; however, commodity distinction is required.

In the N-commodity formulation, a commodity is the ensemble of inputs to the network which are destined to a particular node. The conservation equations are written at each node j for each commodity k as

$$\sum_{\substack{i \text{ such that} \\ OR(i) = j}} f_i^k - \sum_{\substack{i \text{ such that} \\ DS(i) = j}} f_i^k = \begin{cases} r_{jk} & \text{if } j \neq k \\ -\sum_{\substack{m=1 \\ m \neq k}}^{N} r_{mk} & \text{if } j = k \end{cases}$$

where f_i^k is the flow in arc i which is destined for node k. Collecting the equations for each commodity, employing matrix notation, and adding the nonnegativity constraints, the result is denoted

$$Ef^k = b^k, \qquad f^k \geq 0 \tag{1.1.2}$$

where E has dimension N x NA and its elements are ± 1 or 0, $f^k = [f_1^k, f_2^k, \cdots, f_{NA}^k]'$, and $b^k = [b_1^k, b_2^k, \cdots, b_N^k]'$, where $b_j^k = -\sum_{m=1}^{N} r_{mk}$ if $j = k$, and $b_j^k = r_{jk}$ if $j \neq k$. $E = [e_{ij}]$ is generally called the node-arc incidence matrix and is independent of the destination node, k, and $e_{ij} = 1$ if $OR(j) = i$, $e_{ij} = -1$ if $DS(j) = i$, and $e_{ij} = 0$ otherwise. The set F is also defined by 1.1.2; in particular, $f \in F$ if and only if there exist $f^1, f^2, \cdots, f^N$ such that $f = \sum_{k=1}^{N} f^k$ and f^k satisfies 1.1.2 for every $k \in \{1, 2, \cdots, N\}$ [ASSA75].

The N^2-commodity formulation (actually there are $N^2 - N$ commodities) is closely related to the N-commodity formulation except that a commodity is associated with both an origin and a destination. The conservation equations are written at each node j for each commodity as

$$\sum_{\substack{i \text{ such that} \\ OR(i) = j}} f_i^{mk} - \sum_{\substack{i \text{ such that} \\ DS(i) = j}} f_i^{mk} = \begin{cases} r_{mk} & \text{if } j = m \\ -r_{mk} & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

where f_i^{mk} is the flow in arc i with origin m and destination k. Collecting the equations for each commodity, employing matrix notation, and adding the nonnegativity constraints, the result is denoted

$$Ef^{mk} = b^{mk}, \qquad f^{mk} \geq 0 \tag{1.1.3}$$

where E is the same matrix as in 1.1.2, $f^{mk} = [f_1^{mk}, f_2^{mk}, \cdots, f_{NA}^{mk}]'$, and $b^{mk} = [b_1^{mk}, b_2^{mk}, \cdots, b_N^{mk}]'$, where $b_j^{mk} = r_{mk}$ if $j = m$, $b_j^{mk} =$

$-r_{mk}$ if $j = k$, and $b_j^{mk} = 0$ otherwise.

1.2 Extremal Flows, Trees, and Chains

This section will investigate the nature of solutions to linear programs when the constraints have the form of 1.1.1, 1.1.2, or 1.1.3. Depending on the particular format, solutions will be called extremal flows, trees, or chains. The powerful decomposition methods of later chapters will be explained in terms of these notions. The problem now under consideration is to minimize $\sum_{i=1}^{NA} c_i f_i \triangleq c'f$ such that $f \epsilon F$. This linear program, also called a shortest route problem, is a subproblem of the algorithms which employ the decomposition methods. The method of attack will be to solve the problem subject to 1.1.3 for each of the commodoties; superposition of appropriate solutions yields the N-commodity solutions, which in turn are superimposed to yield the solution without commodity distinction.

Consider the following problem:

$$\min c'f \text{ such that } Ef^{mk} = b^{mk}, \quad f^{mk} \geq 0 \qquad (1.2.1)$$

Generally the solution is not bounded since the set F is not bounded. However, in the case that $c \geq 0$ (this assumption is implicit in the remainder of this section), then a bounded solution to 1.2.1 does exist. Assume that commodity (m, k) obeys a pure routing strategy which can be described by a chain, $(m = n_o, n_1, \cdots, n_j = k)$, or as a sequence of arcs, $(i_1, i_2, \cdots, i_j)$, such that $OR(i_p) = n_{p-1}$ and $DS(i_p) = n_p$ for $p \epsilon \{1, 2, \cdots, j\}$. The corresponding flow, $f_i^{mk} = \begin{cases} r_{mk} & \text{if } i \epsilon \{i_1, i_2, \cdots i_j\} \\ 0 & \text{otherwise} \end{cases}$, clearly satisfies 1.1.3, and the cost of such a solution is

$r_{mk}\ (\sum_{p=1}^{j} c_{i_p})$. If c_i is thought of as the distance of arc i, and the length of a chain is the sum of the distances of its corresponding arcs, then minimization of the cost of a pure routing strategy from node m to node k can be seen as equivalent to finding the minimum length chain from node m to node k. If the minimum length chain is unique, then the corresponding flow, denoted φ^{mk}, is the unique solution to 1.2.1. If there are j minimum length chains, whose corresponding flow vectors are denoted $\varphi^{mk}_{(1)}, \cdots, \varphi^{mk}_{(j)}$, then any convex combination, f^{mk}, of the chains is a solution to 1.2.1; for this case, $f^{mk} = \sum_{i=1}^{j} \lambda_i \varphi^{mk}_{(i)}$ such that $\lambda_i \geq 0$ and $\sum_{i=1}^{j} \lambda_i = 1$.

The ambiguity that arises from degeneracy, i.e. more than one minimum length path, needs to be resolved. Later in this thesis, reference will be made to the set of chains which solve 1.2.1 for a given metric c. This set contains exactly one chain for each commodity. Degeneracy is allowed to be resolved arbitrarily as long as one is consistent. Consistency in this matter means that if the declared shortest chain between node m and node k is $(m = n_o, n_1, \cdots, n_j = k)$, then the declared shortest chain between node q and node k where $q = n_p$ for $p \in \{1, 2, \cdots, j-1\}$ must be $(q = n_p, n_{p-1}, \cdots, n_j = k)$.

At this point, one should be aware of the reasons that the solution to 1.2.1 is not bounded in general. If the length of a closed path is negative, then an arbitrarily large flow may circulate on that closed path while still satisfying 1.1.1. Therefore, in this case, the objective, c'f, has no finite lower bound. However, by specifying that the metric, c, must be nonnegative, the existence of a bounded solution

to 1.2.1 is assured.

If one has the capability to solve 1.2.1, then the solution to the following problem is at hand:

$$\text{minimize } c'f \text{ such that } Ef^k = b^k, \quad f^k \geq 0 \qquad (1.2.2)$$

In particular, if one has solved 1.2.1 for every $m \in \{1, 2, \cdots, N\} - \{k\}$, then the solutions may be superimposed to solve 1.2.2. The solution to 1.2.2 is denoted ϕ^k, and ϕ^k is related to the set of chains which solve 1.2.1 by $\phi^k = \sum_{\substack{m=1 \\ m \neq k}}^{N} \varphi^{mk}$. The care in resolving degeneracy is manifested in the resultant form of ϕ^k: the solution to 1.2.2 is called a tree because ϕ^k has exactly N-1 nonzero elements when $r_{mk} > 0$ for every $m \in \{1, 2, \cdots, N\}$ such that $m \neq k$, and the nonzero elements correspond to a set of arcs which is a tree of the network. Each tree could be considered rooted at the destination node with each arc directed towards that node.

Finally, one may minimize $c'f$ subject to 1.1.1 by superimposing the set of chains which solve 1.2.1 or the set of trees which solve 1.2.2. The solution, φ, is called an extremal flow or a shortest route flow, and a relationship for the extremal flow in terms of trees and chains is $\varphi = \sum_{k=1}^{N} \phi^k = \sum_{k=1}^{N} \sum_{\substack{m=1 \\ m \neq k}}^{N} \varphi^{mk}$.

1.3 The Representation of Routing Policies

One relatively compact way to state a routing policy is to provide for each node, m, and each possible destination, k, where $k \neq m$, a set of parameters π_i^{mk} such that $\sum_{i=1}^{NA} \pi_i^{mk} = 1$, $\pi_i^{mk} \geq 0$, and $\pi_i^{mk} = 0$ if

$OR(i) \neq m$. π_i^{mk} is the fraction of flow arriving at node m which is to be sent next along branch i to the node DS(i). The practical importance of such a representation, in the context of packet switched networks, is that the policy requires a small amount of information about any given packet (i.e. the destination) in order for the packet to be processed upon its arrival at any given node.

The ultimate goal of each routing algorithm is to provide such a routing policy, although sometimes this is not evident. Routing policies are often implicit in routing algorithms, and the desired explicit format is quite difficult to extract from the "solution" to a given problem when this situation occurs. These thoughts will be given fuller development in the chapter which follows.

CHAPTER II

NONLINEAR MULTICOMMODITY FLOW PROBLEMS

This chapter deals with the application of nonlinear multicommodity flow models to the determination of a routing policy for a computer-communication network. Kleinrock's model of a packet switched network as a network of queues shall be taken as the foundation of such an application [KLEI64].

The Model. In this model, there exists a computer at each node of the network. The computers may perform a variety of services (in terms of the data processing of packets) which are aimed towards the errorless delivery of messages with minimum delay between the network's subscribers. A long message may be broken into smaller units which are denoted packets. The packets propagate through the network as separate entities and are reassembled at the destination node into the original message. Packets travel through the network in a store-and-forward fashion. That is, each nodal computer observes the destination of every arriving packet. If the packet is destined elsewhere, the computer selects an appropriate outgoing channel and places the packet in the queue associated with that channel to be forwarded in its turn.

The problems associated with packet switched communication networks are numerous, and only a few of these will be mentioned. First is the location of the nodes, the selection of the topology, the choice of the transmission medium, and the selection of capacities.

Error control is not only important on each individual link, but also from source to destination, since packets may be lost for various reasons. Finite storage facilities may require reservation of memory for the reassembly of messages. Descriptions of the state of the network may be desired to allow the network to adapt to failure or heavy loading of nodes and/or links, and then one must decide on how and when to send the network description.

This chapter addresses only one of these problems, namely the routing problem, i.e. how should a nodal computer select the output channel for any given packet which may arrive at its node. This problem is especially important, however, for several reasons. It is related to the topology and capacity selection problem because, to evaluate the relative merits of different selections, one must specify how the network will be used before one can determine what demands will be placed on its various components. Moreover, the routing problem is important for more than just the initial design of the network. It is also important in maintaining the effectiveness of the network during normal operating conditions and during emergency conditions such as failure of components of the network, and in determining how the network should grow with increasing usage. To lose traffic because the network cannot adapt to unusual conditions or to invest capital into a larger network when a more robust routing strategy would suffice are design errors which unnecessarily degrade the cost-effectiveness of the network.

The simplest model of a packet switched network having some

degree of realism is a network of queues model. This approach was first developed by Kleinrock [KLEI64]. All delays are assumed to be negligible except queueing delays which arise from the finite transmission rates over channels (determined by the channel capacities) and the randomness of arrivals of packets which are to be sent over the various channels. One also assumes in this model that each packet must have fully arrived at a node prior to its forwarding, and that the queues associated with the various channels have infinite storage capability. Then the following additional assumptions or approximations are made in order to apply the results of Burke [BURK56] and Jackson [JACK63] concerning networks of queues, making the problem analytically manageable:

(i) The arrival times of packets with an origin i and destination j form a Poisson process with mean arrival rate r_{ij} (bits/second) which is independent of all other network processes.

(ii) The length of a packet, which is proportional to the service time at a node, is an exponentially distributed random variable with mean $1/\mu$ (bits).

(iii) The length of a packet is assumed to be chosen independently of all network processes upon arrival at each successive node from source to destination.

Under all of these assumptions, the average delay of a randomly selected packet, T (seconds/packet), becomes:

$$T = \frac{1}{\mu\gamma} \sum_{i=1}^{NA} \frac{f_i}{C_i - f_i}$$

where f_i is the average bit rate (bits/second) on the ith channel and is determined by the demands and the routing policy, and where $\gamma = \sum_{i,j} r_{ij}$.

The expression for T can become more elaborate if causes for delay other than queueing are included in the model, e.g. nodal processing time, propagation time, network control traffic, etc. [KLEI70]. The methods of this chapter will remain valid as long as the delay, expressed as a function of flow, T(f), is convex and the gradient of T, $\nabla T = [\partial T/\partial f_1, \partial T/\partial f_2, \cdots, \partial T/\partial f_{NA}]'$, is nonnegative over the set $F \cap G$, where $f \in G$ if $0 \leq f_i < C_i$ for every $i \in \{1, 2, \cdots, NA\}$.

Previous Work. One criterion which is used to determine an 'optimal' routing policy is the minimization of the expected delay through the network. Such a routing policy is termed optimal in the static sense with respect to average delay. One can update the policy as one acquires new estimates of the demands, i.e. the set of requirements, $\{r_{ij}\}$. In this case, the algorithms based on the above criterion may be termed quasistatic, but are not necessarily optimal in any sense, although perhaps near optimal in certain senses under certain circumstances.

The algorithms of this chapter are termed centralized, in the sense that the required computations are most efficiently performed at a central location in the network. This of course entails, for an

actual quasistatic implementation, the transfer of some "state of the network" description to a central location, a calculation of a desirable routing policy, and then distribution of this routing information throughout the network. It seems possible that a distributed algorithm may reduce this communication problem at a small cost in nodal computer complexity. The reader is referred to the work of Gallager [GALL77] for an investigation of this possibility, and for other references concerning distributed routing algorithms. The choice of distributed versus centralized routing in an actual implementation probably should be based on many factors such as the reliability of the components of the network and the nature of the demands on the system. Although information must be passed farther, in general, for a centralized procedure, qualitatively one can see that there may exist cases where the information can be passed less often. A distributed algorithm converges to the solution of a particular static problem by the process of information interchange among the nodes. On the other hand, in the case of a centralized algorithm, information is passed only for gathering data and communicating the routing policy to the nodes.

The algorithms to which this chapter will be directed are also referred to as exact rather than heuristic. Exact in this case means that one can come arbitrarily close to the solution of the minimum delay problem using such algorithms, given an exact computing machine and a sufficient number (which is finite) of iterations. Exact algorithms appear to have the same order of complexity as heuristic

algorithms exemplified by those of Frank and Chou [FRAN71a] and Fultz [FULT72]. Moreover, exact algorithms have the advantage of always having the capability of being driven to a near optimal solution, as opposed to heuristic algorithms which invariably seem to perform poorly in some cases.

Furthermore, this chapter is concerned with algorithms which retain the identity of components of the total flows on each link in the solution. Otherwise, a routing policy is not easily derived. An algorithm which performs the minimization without retaining this identity is the Frank-Wolfe (FW) algorithm [FRAN71b]. A very similar algorithm was denoted the flow deviation (FD) algorithm and was initially applied to packet switched communication networks [FRAT73]. The applications of the FW algorithm to transportation networks were studied by Leblanc, Morlock, and Pierskalla [LEBL75] and by Nguyen [NGUY74]. A variation on the FW algorithm was proposed by Klessig [KLES74] who altered the approach to the one-dimensional minimization of the algorithm. Accordingly, the FW algorithm arrives at a value of minimum delay but does not reveal how to achieve this value through routing. One consequence of this is the inability to arbitrarily perturb the demand matrix, and then arrive at the new solution without completely resolving the problem (see Appendix B). Naturally the memory associated with such algorithms is comparatively small, and the FW algorithm can handle quite large networks more easily than algorithms that are capable of determining a routing policy. The use of the FW algorithm is directed towards a

criterion for comparing networks when the network is in its first design stage. The FW algorithm (or equivalently the FD algorithm) was encoded as a part of this thesis in order to serve as a standard of comparison for the computations cited in Chapter IV.

Some multicommodity flow problems have methods which are effective for a small number of commodities. In particular, Golden developed an algorithm for import-export routing involving two commodities [GOLD75], and Schwartz and Cheung developed an algorithm suited to computer-communication networks with a very small number of origin-destination pairs [SCHW76]. The algorithms under consideration in this chapter are directed to those cases where the demand matrix, $R = [r_{ij}]$, has many nonzero terms, although they should not necessarily perform poorly when that is not the case.

The first section of this chapter outlines the algorithm of Cantor and Gerla [CANT74] which is called the EF algorithm, and the remainder is devoted to related algorithms based on alternative decomposition schemes. The object of these algorithms is to minimize T(f) subject to $f \in F$, and to find the associated routing policy. Notice that T(f) only makes sense when f satisfies the capacity constraints, so that $T(f) \triangleq \infty$ if $f_i \geq C_i$ for any $i \in \{1, 2, \cdots, NA\}$.

2.1 The EF Algorithm

The short presentation of this section views the EF algorithm as a column generating programming technique [ASSA75]. The name EF is drawn from the term "extremal flow" which was introduced and

defined in Section 1.2. Initially, one assumes that a restricted set of extremal flows, called a basis $\{\varphi_{(1)}, \varphi_{(2)}, \cdots, \varphi_{(j)}\}$, has been provided. Let f be a convex combination of the elements of the basis, i.e. $f = \sum_{i=1}^{j} \lambda_i \varphi_{(i)}$ such that $\sum_{i=1} \lambda_i = 1$ and $\lambda_i \geq 0$. Clearly, $f \epsilon F$; however, not every element of F can be expressed as a convex combination of the given basis elements. The restricted master problem is

$$\text{minimize } T\left(\sum_{i=1}^{j} \lambda_i \varphi_{(i)}\right)$$

$$\text{such that } \sum_{i=1}^{j} \lambda_i = 1, \quad \lambda_i \geq 0$$

which can be solved as a multidimensional minimization over the parameters $\lambda_1, \lambda_2, \cdots, \lambda_j$ under constraint. Denote the solution to the restricted master as f^*, which is said to be the current estimate of the solution of

$$\text{minimize } T(f) \text{ such that } f \epsilon F \qquad (2.1.1)$$

Since F is a convex set and T(f) is a convex function, then f^* is the solution to 2.1.1 unless there exists $f \epsilon F$ such that

$$\text{Rel Cost}(f) \triangleq \langle \nabla T(f^*), (f - f^*) \rangle < 0 \qquad (2.1.2)$$

where $\langle , \rangle$ denotes the usual inner product in Euclidean space. This condition for optimality is derived from the Kuhn-Tucker theorem [VARA72] by Cantor and Gerla [CANT74]. Define $c = \nabla T(f^*)$; then the minimization of Rel Cost (f) such that $f \epsilon F$ is equivalent to the minimization of c'f subject to 1.1.1. A solution to this problem is an extremal

flow, which is denoted $\varphi_{(j+1)}$, and the method of solution is by solving a shortest route problem as outlined in Section 1.2. This shortest route calculation is called the subproblem of the EF algorithm. If the solution to the subproblem, $\varphi_{(j+1)}$, is such that Rel Cost $(\varphi_{(j+1)}) \not< 0$, then f^* solves 2.1.1 and the algorithm terminates; otherwise, $\varphi_{(j-1)}$ is added to the restricted set of extremal flows, and the master problem is resolved with the assurance that the objective function, T(f), can be lowered, and so forth. The number of extremal flows is finite, and if one can solve the restricted master problem exactly, then it follows that the algorithm solves the problem in a finite number of iterations.

Implicitly, in the above, the first estimate of the solution to 2.1.1 is assumed to satisfy the capacity constraints since the multi-dimensional minimization of the master problem requires a well defined gradient. The starting problem is that of finding such an initial estimate of the solution; this problem is discussed further in Chapter IV where an easily implemented procedure for starting the algorithm is presented.

The analogy of the EF algorithm with linear programming and the simplex method are apparent. The columns of the program are all possible extremal flows. Only a small number of these are known explicitly, namely the members of the basis, which in turn are the only columns with which a nonzero primal variable, λ_i, can be associated. At the conclusion of the master problem (assuming perfect minimization), the cost with respect to the current basis, Rel Cost $(\varphi_{(i)})$, is zero for

every basis element such th... $_i > 0$ (the remainder of the basis elements may be deleted in t... EF algorithm for more efficient use of memory in the computer). Corresponding to the simplex algorithm's search for the nonbasic column with minimum relative cost, the subproblem generates such a column, $\varphi_{(j+1)}$. In the nonlinear problem, Rel Cost $(\varphi_{(j+1)})$ is not only related to the ability of $\varphi_{(j+1)}$ to lower the objective function upon being included in the basis, but also - Rel Cost $(\varphi_{(j+1)})$ is an upper bound to $T(f^*) - T(f^\#)$, where f^* is the current estimate of the optimal solution found by the master problem, and $f^\#$ is the solution to 2.1.1 (see Appendix A, Thm. 2.1A). Theoretically, one stops the algorithm when Rel Cost $(\varphi_{(j+1)}) \leq 0$; however, as a practical matter, the upper bound is a powerful tool in knowing when stopping the algorithm is appropriate.

The multidimensional minimization which is required in the master problem was solved via the gradient projection method by Cantor and Gerla [CANT74]. This technique reduces the problem to a series of one-dimensional minimizations each of which is along a feasible downhill direction (but not the direction of steepest descent). Best employed a variation of this approach towards the multidimensional minimization problem which may be of some benefit [BEST75]; however, there is not an actual comparison showing net reduction of overall computation time due to his alternate approach, even though his algorithm was actually encoded.

This section is concluded with some remarks on finding the optimal routing strategy from the solution to the EF algorithm.

Knowing the optimal flow $f^{\#}$ to the problem 2.1.1 does not imply the solution to the routing policy because $f^{\#}$ is a convex combination of flows without commodity distinction. Specification of an optimal routing policy, however, requires at least a knowledge of the destinations of the flows making up the solution. For example, if $\varphi^{k}_{(i)}$ is the component of $\varphi_{(i)}$ destined for node k, then $f^{\#} = \sum_{i=1}^{j} \lambda_i \varphi_{(i)} = \sum_{i=1}^{j} \sum_{k=1}^{N} \lambda_i \varphi^{k}_{(i)}$ and $f^{k\#} \triangleq \sum_{i=1}^{j} \lambda_i \varphi^{k}_{(i)}$ is the component of the optimal flow destined for node k. Consequently, the routing variables defined in Section 1.3 may be calculated as follows:

$$\Pi_i^{mk} = \frac{f_i^{k\#}}{\sum_{\substack{j \text{ such that} \\ OR(j) = m}} f_j^{k\#}} \qquad (2.1.3)$$

Clearly, the optimal routing policy is known only if sufficient information is stored with each extremal flow to be able to reduce it into its corresponding trees. The most compact way to retain this information (and the way suggested by Cantor and Gerla) is via a routing matrix, NX, of dimension N x N where NX_{ij} is the next arc from node i on the shortest route to node j. Thus, having stored a routing matrix for each extremal flow, when the EF algorithm terminates, the necessary trees may be constructed which allow solution of 2.1.3 for every pair of m and k.

2.2 The TR Algorithm

The TR algorithm is an original variation on the Cantor and Gerla routing algorithm with a decomposition based on trees rather

than extremal flows. The computational experience with the TR algorithm which will be presented in Chapter IV and the theoretical comparisons in the remainder of this chapter suggest that the TR routing algorithm should be competitive in speed with the EF algorithm in solving 2.1.1, and makes more efficient use of memory in a computer implementation. Moreover, an optimal routing policy is determined at the conclusion of the algorithm with far less effort.

This is the master problem for the TR algorithm:

$$\text{minimize } T(f) = T\left(\sum_{k=1}^{N} \sum_{i=1}^{t_k} \lambda_{ik} \varphi_{(i)}^{k}\right)$$
$$\text{such that } \sum_{i=1}^{t_k} \lambda_{ik} = 1, \quad \lambda_{ik} \geq 0 \tag{2.2.1}$$

where each $\varphi_{(i)}^{k}$ is a member of a given restricted set of trees (as defined in Section 1.2) which is called a basis, and t_k denotes the number of trees in the basis associated with flows destined for node k. In 2.2.1, f is said to be a convex combination of the trees in the given basis. Denote the solution to 2.2.1 as f^*.

The subproblem to the TR algorithm is analogous to that of the EF algorithm: find the set of trees which minimizes c'f subject to 1.1.2 where $c = \nabla T(f^*)$. If f^* is not the global optimum, then newly generated trees are added to the existing basis and the algorithm is iterated.

At first glance, this master problem may look more difficult to solve than the master problem of the EF algorithm due to the increase of the number of columns and associated variables. On the other hand,

an advantage of the format of 2.2.1 is apparent. Suppose a restricted set of trees was generated by the same shortest route problems which generated a restricted set of extremal flows. In this case, the region of F which can be expressed as a convex combination of the trees includes the region which can be expressed as a convex combination of the extremal flows. If the respective master problems were solved at this point, then the TR master problem would generally find a better estimate of the solution to 2.1.1.

Section 2.3 will deal more precisely with the comparisons of the master problems; the remainder of this section compares memory requirements and ease in reducing solutions to optimal routing policies. Cantor and Gerla showed that the solution to 2.1.1 can be expressed as a convex combination of at most NA+1 extremal flows [CANT74]. Via virtually identical arguments, the solution to 2.1.1 can be expressed as a convex combination of at most NA+N trees (see Appendix A, Thm. 2.2.A). Consider the storage implications of these facts (it will be assumed that both integers and reals require one memory location, although on some machines integers may be stored more compactly than reals). Trees have no more than N-1 nonzero terms, and if they are stored along with an appropriate arc index for each term, then the maximum storage requirement is 2(N-1)(NA+N) locations for the basis. The corresponding maximum requirement for the EF basis is NA(NA+1). If the number of arcs exceeds three times the number of nodes, then the former term is smaller.

Now suppose that the optimal routing policy is desired at the

conclusion of the algorithm, so that one must also store a routing matrix for each extremal point. The maximum memory requirement associated with the EF basis is now $(N^2+NA)(NA+1)$ which is about a factor of N greater than the required storage for the TR algorithm. In order to find the optimal routing policy, one must find $f^{k\#}$ for every $k\epsilon\{1,2,\cdots,N\}$. For the TR algorithm, this requires at most $(N-1)(NA+N)$ multiplications in a very straightforward procedure. After one has found the trees associated with all the extremal flows in the solution of the EF algorithm, combining them to form every $f^{k\#}$ requires a maximum of $(N-1)(NA+1)(N)$ multiplications and additions.

Evidently, the TR algorithm looks very attractive in both storage requirements and convenience of the format of the solution for establishing the optimal routing policy. Since the subproblem of the EF algorithm is of exactly the same complexity as that of the TR algorithm, only a comparison of the efficiency of the master problems remains to be examined. The next section deals with a suggested method for solving the multidimensional minimization of the TR master problem and discusses the complexity in a comparative manner with that of the EF master problem.

2.3 The TR Master Problem

Several concepts presented in Section 2.1 will be generalized in order that the TR algorithm's master problem may be effectively discussed. Initially, the current estimate of the solution, f^*, is assumed to exist from a previous TR restricted master problem solution. Because of the format of the TR algorithm, f^* may be

expressed as a sum of N-commodity flows, i.e. $f^* \triangleq \sum_{k=1}^{N} f^{k*}$. The respective subproblem generates the set of trees, $\{\varphi^1, \varphi^2, \cdots, \varphi^N\}$, which minimizes

$$\text{Rel Cost}(f^k) \triangleq \langle T(f^*),\ (f^k - f^{k*}) \rangle \qquad (2.3.1)$$

$$\text{for every } k \in \{1, 2, \cdots, N\}$$

Clearly, Rel Cost $(\varphi^k) \leq 0$; and if Rel Cost $(\varphi^k) = 0$ for every $k \in \{1, 2, \cdots, N\}$, then f^* is not only the restricted optimal but the global optimal.

The relative cost of a tree is related to its ability to reduce the objective, T(f), should it be included in the basis. Assuming now that the new trees with negative relative costs have been added to the basis, the new TR master problem requires a multidimensional minimization. Specifically, minimize T(f) such that $f = \sum_{k=1}^{N} \sum_{i=1}^{t_k} \lambda_{ik} \varphi_{(i)}^k$ where $\sum_{i=1}^{t_k} \lambda_{ik} = 1$. For simplicity, it is assumed that this is done via a minor alteration on the gradient projection method which Cantor and Gerla use for the EF algorithm [CANT74]. Calculate $\partial T(f^*)/\partial \lambda_{ik} = \langle \nabla T(f^*), \varphi_{(i)}^k \rangle$ for every tree in the basis. Define $\Delta\lambda_{ik} = -\partial T(f^*)/\partial \lambda_{ik} + (1/t_k) \cdot \sum_{m=1}^{t_k} \partial T(f^*)/\partial \lambda_{mk}$. Then it is easy to show that the direction, Δf, defined by $\Delta f = \sum_{k=1}^{N} \sum_{i=1}^{t_k} \Delta\lambda_{ik} \varphi_{(i)}^k$ is a feasible downhill direction from f^* (this assumes $\lambda_{ik} > 0$ unless λ_{ik} is associated with a tree which has just been added to the basis; see Appendix A, Thm. 2.3.A). The condition $\lambda_{ik} > 0$ is maintained by dropping any tree from the basis if ifs associated λ_{ik} is driven to zero in the process of the multi-dimensional minimization.

The next step is to solve the one-dimensional minimization problem: $\underset{0 < \epsilon < \epsilon^*}{\text{minimize}}\ T(f^* + \epsilon \Delta f)$ where $\epsilon^* = \underset{\Delta\lambda_{ik} < 0}{\text{minimum}}\ (-\lambda_{ik}/\Delta\lambda_{ik})$. For the optimal value of ϵ, denoted $\hat{\epsilon}$, $f^* \leftarrow f^* + \hat{\epsilon}\Delta f$ and $\lambda_{ik} \leftarrow \lambda_{ik} + \hat{\epsilon}\Delta\lambda_{ik}$. A new feasible downhill direction may be found at this point, and the procedure is repeated until one is content with the current f^*.

The one-dimensional minimization requires about the same amount of effort whether it is a part of the TR master problem or of the EF master problem. Moreover, the computational effort of finding Δf is dominated by that of finding $\langle \nabla T(f^*), \varphi^k_{(i)} \rangle$ for every tree, and then forming the sum, $\Delta f = -\sum_{k=1}^{N} \sum_{i=1}^{t_k} \Delta\lambda_{ik}\, \varphi^k_{(i)}$. If t is the number of trees in the basis, then the number of multiplications and additions for the TR master problem is about $2t(N-1)$. A similar analysis for finding Δf in the EF master problem results in $2e(NA)$ multiplications and additions where e is the number of extremal flows in the basis. The efforts involved in finding Δf seem comparable, although they clearly would vary between particular problems.

For the number of trees in the basis not to grow out of hand, the algorithm must remove from the basis about as many trees in the master problem as are added from the previous subproblem. Typically, only one tree may have its associated λ_{ik} driven to zero for each feasible downhill direction, Δf, which is found. Thus, on the average only a small number of trees with the most negative costs should be added to the basis after each subproblem.

As it turns out in the actual operation of the algorithm, the potential problem of unbounded growth of the basis can be avoided. In

the same way that one bounds the absolute error in any estimate of the solution, one can bound the reduction in the objective that could be achieved by an exact multidimensional minimization with the current basis. Specifically, if

$$g^k \triangleq \underset{1 \le i \le t_k}{\text{minimum}} \text{ Rel Cost } (\varphi^k_{(i)})$$

then $-\sum_{k=1}^{N} g^k$ bounds the error due to inexact minimization. Now if this bound is nearly as large as the absolute bound, it is clear that the utility of adding columns is small and more effort should be spent on the master problem rather than in generating columns. From the experience gained with this algorithm, it is a good policy to add columns only when the ratio of the maximum error due to inexact multidimensional minimization to the absolute error is under some prespecified threshold. Under this mode of operation, the algorithm rapidly reaches the point where the rate of adding new columns is close to the rate at which columns are being evicted.

In summary, the overall memory requirements and the convenience of format of the TR algorithm would cause one to favor it over the EF algorithm. The relative speed of the master problems is difficult to predict, although it has been shown that the efforts involved in performing related tasks within the master problems are roughly equivalent.

2.4 The CH Algorithm

The CH algorithm is a further extension of the TR algorithm and

the EF algorithm in which the decomposition is based on chains rather than trees or extremal flows. The comparative discussion in this section plus the computational experience cited in Chapter IV suggest that the CH algorithm is competitive with respect to computation time with the EF and TR algorithms, and is competitive in terms of memory requirements with the TR algorithm.

This is the master problem of the CH algorithm:

$$\text{minimize } T(f) = T\left(\sum_{m=1}^{N} \sum_{\substack{k=1 \\ k\neq m}}^{N} \sum_{i=1}^{c_{mk}} \lambda_{imk}\, \varphi_{(i)}^{mk}\right) \tag{2.4.1}$$

$$\text{such that } \sum_{i=1}^{c_{mk}} \lambda_{imk} = 1, \quad \lambda_{ik} \geq 0$$

where each $\varphi_{(i)}^{mk}$ is a member of the given basis consisting of chains (as defined in Section 1.2), and c_{mk} is the number of chains in the basis having origin m and destination k. In 2.4.1, f is said to be a convex combination of the chains in the basis. Again, denote the solution of the master problem as f^*.

The subproblem of the CH algorithm is analogous to the previous subproblems, namely: find the set of chains which minimize $\langle \nabla T(f^*), f \rangle$ subject to 1.1.3. If it is found that f^* is not the global optimum, then the newly generated chains are added to the existing basis and the algorithm is iterated.

Although once again the master problem has been complicated, one hopes for a pay-off due to additional flexibility in the minimization. One may notice that this algorithm assumes the flavor of others which exist in the literature, in particular that of Dafermos [DAFE71]. The

principal difference between them is in the form of the downhill directions. Using a methodology similar to that of the TR and EF algorithms in the master problem, the CH algorithm finds a sequence of downhill directions, Δf, and each generally alters the flow in every link of the network; however, for each Δf in the CH algorithm, there corresponds N(N-1) downhill directions in the Dafermos algorithm, each of which is associated with an origin-destination pair and a transfer of flow from the currently worst chain in terms of marginal cost to that of the best. Dafermos' discussion of her algorithm presumed knowledge of every chain in the network; however, clearly a chain generating scheme can be, in fact must be, applied for networks of any size.

Qualitatively, the Dafermos type of scheme wastes a great deal of effort when the algorithm is "far" from a solution. One is "fine tuning" portions of the network when the flow in that area may later be drastically different due to overall considerations; however, when the entire network is close to the solution, this sort of sequential fine tuning may be effective. The most likely meaning of the word 'far' is dictated by the point of the occurrence of the tailing off phenomenon (which is well known for the FW algorithm and also seems to exist for the EF, TR and CH algorithms which are no-so-distant relatives). Tailing off refers to a decrease in experimental convergence rate as the algorithm proceeds. Although the point of this occurrence is a function of many factors, such as the demands on the network, the form of the objective function, and the network topology, onset when the

objective reaches from 1 to 2% of optimal is typical.

In terms of comparison, one can show that a maximum of (NA+N(N-1)) chains is required to express the solution to 2.1.1 (see Appendix A, Thm. 2.4.A). Chains have a maximum of (N-1) nonzero terms, each of which requires only the storage of the arc number. This means that (N-1)(NA+N(N-1)) bounds the memory requirements for the basis of the CH algorithm, as compared to 2(N-1)(NA+N) for the TR algorithm. The latter term is smaller in the case that each node, on the average, has less than (N-2) nodes emanating from it. Clearly, the bound on the number of nonzero elements in the chains is quite liberal. In fact, as the network tends towards being fully connected, the two bounds above approach one another; and, since chains in general require even smaller storage areas than trees, the CH algorithm memory requirement becomes quite small compared to the TR algorithm. The possible advantages, however, are somewhat reduced by other effects as: (i) an N-fold increase in the number of identified commodities and the memory associated with each, (ii) storage of variable length chains requires some overhead in storage or computation time since lengths are unpredictable, (iii) storage is further complicated since one desires the basis to be fluid, i.e. to allow insertion and removal of chains at will, and (iv) the ratio of the minimum number of chains in a solution to the maximum number of chains in a solution is comparatively large, so that even when the solution is known to be relatively simple, a large percentage of the maximal memory is still required.

The extension of the TR master problem to that of the CH master problem is reasonably straightforward, and is therefore left to the reader. As in comparing the computational efforts of the EF algorithm to the TR algorithm where analogous portions of the algorithms are normally calculated with the same ease, the fundamental question in comparing the TR algorithm to the CH algorithm is that of the relative efforts of calculating Δf. For the CH algorithm, this effort is dominated by: (i) calculation of $\langle \nabla T(f^*), \varphi_{(i)}^{mk} \rangle$ for every chain, which requires one multiplication and as many additions as there are nonzero elements of $\varphi_{(i)}^{mk}$, (ii) calculation of the $\Delta\lambda_{imk} = -\partial T(f^*)/\partial\lambda_{imk} + (1/c_{mk}) \sum_{j=1}^{c_{mk}} \partial T(f^*)/\partial\lambda_{imk}$, which requires one division for each commodity and as many additions as basis elements, and (iii) formation of the sum $\Delta f = -\sum_{k=1}^{N} \sum_{\substack{m=1 \\ m\neq k}}^{N} \sum_{i=1}^{c_{mk}} \Delta\lambda_{imk} \varphi_{(i)}^{mk}$ which for any given chain requires one multiplication and as many additions as nonzero elements of the chain. If c is the number of chains and $\bar{n}$ is the average number of links in a chain, then the above sums to 2c multiplications and N^2 divisions and $2c\bar{n}+N^2$ additions. This does not seem substantially different from the $2t(N-1)$ multiplications and additions of the TR algorithm.

2.5 Transitional Remarks

A few remarks are in order before concluding this chapter. The next chapter deals with shortest route algorithms which are the tools for solving the subproblems in an efficient manner. The effort associated with solving the subproblems is nearly equivalent to that of

solving the master problems [CANT74], and decomposition techniques are suggested for reducing shortest route computation time for certain classes of networks. Chapter IV discusses programs which solve network problems with linear objectives which use the same decomposition techniques of the EF, TR, and CH algorithms, and the computational experience of that chapter will be seen to support the general conclusions of this chapter.

CHAPTER III

SHORTEST ROUTE ALGORITHMS

The problem of finding all the shortest routes in a directed network has an extensive literature [DREY69, PIER75] due to the number of network problems to which shortest route algorithms are applied. This chapter presents two new shortest route algorithms which can significantly reduce the required computation time when the network is less than fully connected. These algorithms are intended for use where many shortest route computations will be performed for the same topology. The first is based on original decomposition ideas and is called the node-by-node decomposition (NXN) algorithm.† The second is based on Hu's decomposition algorithm [HU71, HU69, YEN71] and is designated the improved Hu (IHU) algorithm.

The shortest route problem is formulated as a shortest distance problem where $D = [d_{ij}]$ is a given matrix. The number d_{ij} represents the length of the directed arc from node i to node j, and thus it is assumed $d_{ii} = 0$. In this chapter, one further assumes that the length assigned to an arc is ∞ if and only if the arc does not exist in some sense. A path P from i to j is an ordered sequence $i = k_o, k_1, \cdots, k_{m-1}, k_m = j$, and the length of the path, L(P), is defined as $L(P) = \sum_{r=1}^{m} d_{k_{r-1}k_r}$. If P is any closed path, then it is assumed $L(P) \geq 0$ so that the shortest distance problem is well defined. Then the

---

† After completion of this work, the equivalence of the NXN algorithm with previous work done at Network Analysis Corporation under ARPA Order No. 1523 [NAC71] was discovered.

problem is to find $D^* = [d^*_{ij}]$ where $d^*_{ij} = \min L(P)$ for P ranging over all paths from i to j. Knowing D^* alone does not specify the shortest routes, but it is a well documented fact that by appropriate bookkeeping as one calculates D^*, the shortest routes can also be established [HU71]. Additional discussion of the problem of keeping track of the shortest routes is included in Section 3.6.

Typically D^* is calculated as a series of refinements on D. Floyd's algorithm [FLOY62] is cited for an N node network:

For every $i \in \{1, 2, \cdots, N\}$, do step a:

(a) For every $j, k \in \{1, 2, \cdots, N\}$, do step b:

(b) $d_{jk} \leftarrow \min (d_{jk}, d_{ji} + d_{ik})$

where "$\leftarrow$" means "is replaced by". The algorithm requires N^3 additions and N^3 comparisons, and it is generally assumed additions and comparisons take about the same amount of time so that one says Floyd's algorithm requires $2N^3$ operations. At the conclusion of the algorithm D^* has replaced D. Proof of the algorithm is found elsewhere [HU69], but the interested reader can easily convince himself that when i has been stepped from 1 through i_o then the current value of d_{jk} is the minimal distance over all paths from j to k under the condition that the intermediate nodes are elements of the set $\{1, 2, \cdots, i_o\}$.

No algorithm which solves the shortest route algorithm could be any simpler to encode, but there are a variety of faster algorithms in terms of number of operations [HOFF76, SPIR73]. The standard

against which the new decomposition algorithms will be measured is Yen's implementation of Dijkstra's algorithm [DIJK59, YEN71] requiring $\frac{3}{2}N^3$ operations. Those algorithms cited above claiming even fewer operations are not significantly faster, theoretically, for networks of the size for which computational experience is cited in this paper; furthermore, some of the apparent gains of the theoretically faster algorithms would be offset by their additional algorithmic complexity.

3.1 The NXN Algorithm

The NXN algorithm for solving the shortest route problem is actually a special case of the following new $2N^3$ operation algorithm:

(1) For every $i \in \{1, 2, \cdots, N-2\}$ in order, do step a:

  (a) For every $j, k \in \{i+1, i+2, \cdots, N\}$, do step b:

    (b) $d_{jk} \leftarrow \min (d_{jk}, d_{ji} + d_{ik})$

(2) For every $i \in \{N-2, N-3, \cdots, 1\}$ in order, do step a:

  (a) For every $j, k \in \{i+1, i+2, \cdots, N\}$, do steps b and c:

    (b) $d_{ij} \leftarrow \min (d_{ik} + d_{kj}, d_{ij})$

    (c) $d_{ji} \leftarrow \min (d_{jk} + d_{ki}, d_{ji})$

An intuitive proof of this algorithm will be helpful in understanding the NXN algorithm. By inductive reasoning similar to that for Floyd's algorithm, when step 1 has been completed for $i = i_o$, then d_{jk} (for $j, k > i_o$) represents the conditional shortest j to k distance subject to all intermediate nodes being elements of the set

$\{1, 2, \cdots, i_o\}$. Consequently, when step 1 has been completed, then the d_{jk} (for $j, k > N-2$) represent unconditional shortest distances d^*_{ij}.

Note that an arbitrary i to j path (for $j > i$) must be of the form $i, \cdots, r, \cdots, j$, where r is the first element in the path such that $r > i$; and, if this path is the shortest path, then its length is $d^*_{ir} + d^*_{rj}$. When performing step 2 for $i = N-2$, d^*_{rj} is known and d^*_{ir} must be the same as the minimal d_{ir} conditional on all intermediate nodes being elements of the set $\{1, 2, \cdots, N-3\}$; it follows that at the end of step 2 for $i = N-2$, $d_{ij} = d^*_{ij}$, and similarly $d_{ji} = d^*_{ji}$ for every $j \epsilon \{N-1, N\}$. Clearly, inductive reasoning shows that at the end of the algorithm $D = D^*$.

The NXN algorithm will now be presented. However, in order to simplify the discussion, it is assumed that all of the arcs are duplex, i.e. if $d_{ij} < \infty$, then $d_{ji} < \infty$. Define C_i, called the ith connection set, as follows: $j \epsilon C_i$ if $j > i$ and there exists a path P from i to j such that $L(P) < \infty$ and every intermediate node k satisfies $k < i$. Notice that the C_i are functions of topology only.

In step 1 of the above algorithm, $d_{ji} = \infty$ if $j \notin C_i$ and $d_{ik} = \infty$ if $k \notin C_i$. Furthermore, in step 2 of the above algorithm, $d_{ik} = \infty$ and $d_{ki} = \infty$ if $k \notin C_i$. The corresponding operations are clearly unnecessary; the algorithm obtained by deleting them is called the NXN algorithm:

(1) For every $i \epsilon \{1, 2, \cdots, N-2\}$ in order, do step a:

(a) For every $j, k \epsilon C_i$, do step b:

(b) $d_{jk} \leftarrow \min (d_{jk}, d_{ji} + d_{ik})$

(2) For every $i \epsilon \{N-2, N-3, \cdots, 1\}$ in order, do step a:

(a) For every $j \epsilon \{i+1, i+2, \cdots, N\}$ and $k \epsilon C_i$, do steps b and c:

(b) $d_{ij} \leftarrow \min (d_{ik} + d_{kj}, d_{ij})$

(c) $d_{ji} \leftarrow \min (d_{jk} + d_{ki}, d_{ji})$

A decomposition is defined as an ordering of the nodes. Since the connection sets are a function of the decomposition, the number of operations which the algorithm requires is also a function of the decomposition, as will be demonstrated in the following section.

In the case where some of the arcs are not duplex, two alternatives are available. (For data communication networks, half duplex links are rare because of the resulting complications in error control.) The first is to change the definition of C_i as follows: $j \epsilon C_i$ if $j > i$ and there exists a path P from i to j or from j to i such that $L(P) < \infty$ and every intermediate node k satisfies $k < i$. This approach causes unnecessary operations for the algorithm. The alternative is to define two connection sets for each node--one for the incoming connections and one for the outgoing connections. In the latter case, one must alter the NXN algorithm to incorporate the efficiencies of the additional connection sets. The increased algorithmic complexity of the second approach and the resultant additional computer steps

must be weighed against the number of unnecessary operations of the first approach for the problem at hand.

3.2 Decomposing the Network for the NXN Algorithm

This section is introduced via an example. Consider Figs. 3.2.1 and 3.2.2 in which the same network has been decomposed two ways. For the first, $C_i = \{i+1, N-1, N\}$ when $i \in \{1, 2, \cdots, N-3\}$ and $C_{N-2} = \{N-1, N\}$; the number of operations for the NXN algorithm is calculated in a straightforward fashion as:

$$\text{Step 1,} \qquad \left(\sum_{i=1}^{N-3} (2)\ (3)\ (3)\right) + (2)\ (2)\ (2)$$

$$\text{Step 2,} \qquad \left(\sum_{i=1}^{N-3} (2)\ (2)\ (3)\ (N-i)\right) + (2)\ (2)\ (2)\ (2)$$

which totals $6N^2 + 12N - 66$. By contrast, for the decomposition of Fig. 3.2.2 $C_i = \{i+1, i+2, \cdots, N\}$ which is exactly the same as if the network was fully connected, and it follows immediately that the NXN algorithm requires $2N^3$ operations. This example makes it clear that the choice of decomposition can have a profound effect on the efficiency of the algorithm.

For an arbitrary network, finding the optimal decomposition in the sense of minimizing the required number of operations for the NXN algorithm is not a trivial problem and probably can only be solved by exhaustive comparison. The method of choosing the decomposition for the examples which are presented later in Section 3.5 deviated only slightly from the following heuristic procedure:

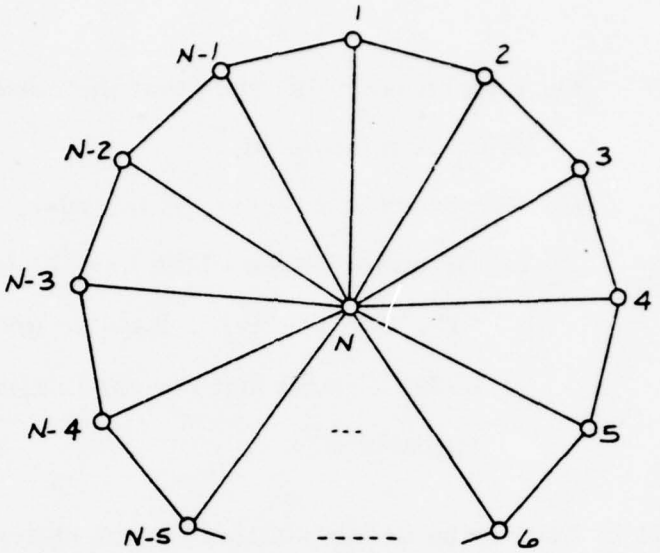


Fig. 3.2.1 An N node network with an NXN decomposition implied by the numbering of the nodes.

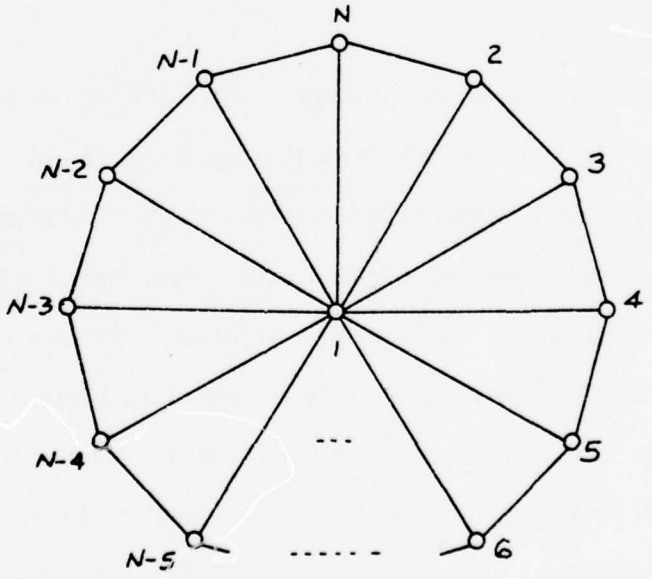


Fig. 3.2.2 The same N node network as in Fig. 3.2.1 with a distinct NXN decomposition.

(1) Label a node "1" such that the cardinality of C_1 is minimized.

(2) For every $i \in \{2, 3, \cdots, N\}$ in order, do step a:

(a) Given the nodes which have been labeled "1", "2", $\cdots$, "i-1", label an unlabeled node "i" such that the cardinality of C_i is minimized.

The effort in finding the decomposition via the above procedure is on the same order as doing a shortest route computation via Floyd's algorithm, and as a consequence computer time savings are realized only when the NXN algorithm is iterated several times for the same topology.

There are a large number of networks such that the computation time does not vary widely with the decomposition. Such networks could be termed "locally connected" and have the property that the nodes to which there are direct arcs from any given node are very likely to have direct arcs to one another. In this case, the nodes could be numbered very rapidly by eye with little degradation in efficiency (nodes must at some point in time be assigned a number anyhow in order to communicate the topology to the computer), and in the first shortest route computation the connection sets could be established with very little effort. In fact, the only modification to the NXN algorithm is an additional step which is included just before step 1a:

(aa) Initially $C_i = \emptyset$; for $j \in \{i+1, i+2, \cdots, N\}$, do step (bb):

(bb) Include j in C_i if $d_{ij} < \infty$.

The additional operations required by this step number $\frac{1}{2}N^2$, which is quite modest for the potential gains.

3.3 The IHU Algorithm

The presentation of the IHU algorithm requires some additional definitions. For this algorithm, a network decomposition is defined as a division of the network's nodes into ordered subsets $S_1, S_2, \cdots, S_k$ such that for every $i \in S_m$ and $j \in S_\ell$, $d_{ij} = \infty$ if $|m-\ell| > 1$. Every node of the network belongs to exactly one subset. The submatrix $D_{S_i S_m}$ contains all the distances of arcs from elements of S_i to elements of S_m, and its dimensionality is $|S_i| \times |S_m|$ (where $|S|$ means the cardinality of set S). Evidently, $D_{S_i S_m}$ has no finite entries in the case $|i-m| > 1$ (see Fig. 3.3.1).

Various matrix operations will be performed on the submatrices to generate the desired shortest distance matrix. Let $D_{S_i S_i} \leftarrow \xi D_{S_i S_i}$ mean $D_{S_i S_i}$ is replaced by the shortest distance matrix computed from the submatrix $D_{S_i S_i}$. Define $A \cdot B = [\min_m (a_{im} + b_{mj})]$ and $\min(A, B) = [\min(a_{ij}, b_{ij})]$. Also let $S_1 \cup S_2 \cup \cdots \cup S_m = \Omega_m$, and if $m = k$ (where k is the number of ordered sets) then $\Omega_m = \Omega_k \triangleq \Omega$. Define the conditional shortest distance submatrix, $D^*_{S_i S_i}(\Omega_m)$, as the shortest distance submatrix under the restriction that all the intermediate nodes on the respective conditional shortest

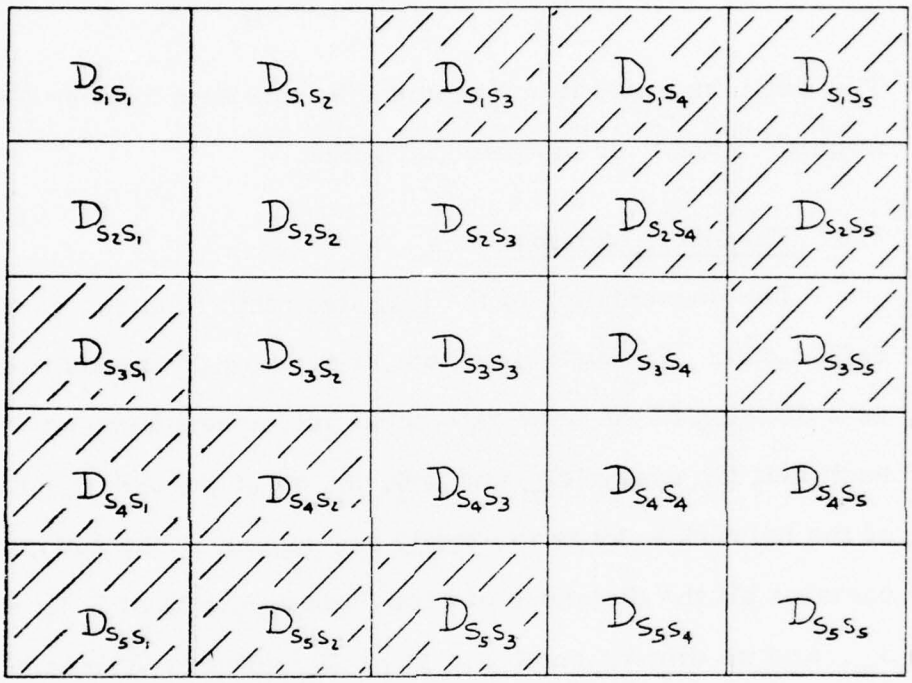


Fig. 3.3.1 The form of the D matrix for the IHU algorithm in the case k = 5. If the decomposition is to be acceptable, the shaded submatrices have no finite entries prior to the algorithmic operations on the D matrix.

routes are members of Ω_m. In the case m = k, $D^*_{S_iS_j}(\Omega_m) = D^*_{S_iS_j}(\Omega) \triangleq D^*_{S_iS_j}$.

Under the assumption that an allowable decomposition has been given, the following algorithm generates all the shortest distances in the network (the parenthetical equality to the right of each step is the claim of what each step accomplishes):

(1) $D_{S_1S_1} \leftarrow \xi D_{S_1S_1}$ $\quad (= D^*_{S_1S_1}(\Omega_1))$

(2) For every $i \in \{1, 2, \cdots, k-1\}$ in order, do steps a, b, c, and d:

(a) $D_{S_{i+1}S_i} \leftarrow D_{S_{i+1}S_i} \cdot D_{S_iS_i}$ $\quad (= D^*_{S_{i+1}S_i}(\Omega_i))$

(b) $D_{S_iS_{i+1}} \leftarrow D_{S_iS_i} \cdot D_{S_iS_{i+1}}$ $\quad (= D^*_{S_iS_{i+1}}(\Omega_i))$

(c) $D_{S_{i+1}S_{i+1}} \leftarrow \min (D_{S_{i+1}S_{i+1}}, D_{S_{i+1}S_i} \cdot D_{S_iS_{i+1}})$

$(= D^*_{S_{i+1}S_{i+1}}(\Omega_i))$

(d) $D_{S_{i+1}S_{i+1}} \leftarrow \xi D_{S_{i+1}S_{i+1}}$ $\quad (= D^*_{S_{i+1}S_{i+1}}(\Omega_{i+1}))$

(3) For every $i \in \{k, k-1, \cdots, 3, 2\}$ in order, do steps a, b, and c:

(a) $D_{S_iS_{i-1}} \leftarrow D_{S_iS_i} \cdot D_{S_iS_{i-1}}$ $\quad (= D^*_{S_iS_{i-1}})$

(b) $D_{S_{i-1}S_i} \leftarrow D_{S_iS_{i-1}} \cdot D_{S_iS_i}$ $\quad (= D^*_{S_{i-1}S_i})$

(c) $D_{S_iS_{i-1}} \leftarrow \min (D_{S_{i-1}S_i}, D_{S_{i-1}S_i} D_{S_iS_{i-1}})$

$(= D^*_{S_{i-1}S_{i-1}})$

(4) For every $r \in \{2, 3, \cdots, k-1\}$ in order, do step a:

(a) For every $i, j \in \{1, 2, \cdots, k\}$ if $|i-j| = r$, do step b:

(b) $D_{S_i S_j} \leftarrow D_{S_i S_p} \cdot D_{S_p S_j}$ $\quad (= D^*_{S_i S_j})$

where p is an element of the set $Q = \{s+1, s+2, \cdots, t-2, t-1\}$ for $s = \min(i, j)$ and $t = \max(i, j)$ such that $|S_p| \leq |S_m|$ for every $m \in Q$.

A rigorous proof of the algorithm would be very lengthy and repetitious, and the interested reader is referred to Hu's work [HU69] for exposition of a similar proof. Steps 1 and 2 are bootstrapping successive diagonal and first off-diagonal submatrices, so that at the end of step 2, $D_{S_k S_k} = D^*_{S_k S_k}$. Step 3 is essentially a backwards form of step 2 and replaces the diagonal and first off-diagonal submatrices with the respective unconditional shortest distance submatrices. Step 4 is one method for finding the unconditional shortest distance submatrices corresponding to decomposition sets which are separated by at least one intermediate set. The ordering in step 4 allows p to be any element of the set Q, and the particular choice of p minimizes the number of operations.

If one assumes that the shortest distance calculations for submatrices are done via Floyd's method (requiring $2p^3$ operations for a p x p submatrix) and that the pseudo-multiplications are done in a straightforward manner (requiring 2pqr operations to calculate A·B where A is dimension p x q and B is q x r), then the number of operations required by the IHU algorithm is:

Step 1, $2|S_1|^3$

Step 2, $2\sum_{i=1}^{k-1}(2|S_{i+1}|\,|S_i|^2 + |S_i|\,|S_{i+1}|^2 + |S_{i+1}|^3)$

Step 3, $2\sum_{i=2}^{k}(2|S_i|^2\,|S_{i-1}| + |S_{i-1}|^2\,|S_i|)$

Step 4, $2\sum_{\substack{i,j \\ |i-j|>1}} |S_i|\,|S_j|\,|S_p|$

The total number of operations is then:

$$2\left(\sum_{i=1}^{k-1}|S_i \cup S_{i+1}|^3 - \sum_{i=2}^{k-1}|S|^3 + \sum_{\substack{i,j \\ |i-j|>1}} |S_i|\,|S_j|\,|S_p|\right)$$

One may compare the IHU algorithm to other versions of Hu's algorithm. For any given decomposition, the IHU algorithm requires fewer operations than the fastest version of Hu's algorithm known to the author, which is that due to Yen [YEN71]. For purposes of comparison, an example which commonly appears in the literature [HU71, HU69, YEN71] is presented. Let $|S_i| = \delta$ for i even and $|S_i| = t$ for i odd. Assume $\delta \leq t$, and let k, the number of sets, be odd. Define $m = (k+1)/2$. In this case, the new algorithm requires $2(mt^3 + (m^2 + 5m - 6)t^2\delta + (2m^2 + 2m - 6)t\delta^2 + (m^2 - 4m + 5)\delta^3)$ operations. Yen's modification requires $2(mt^3 + (m^2 + 6m - 7)t^2\delta + (2m^2 + 10m - 20)t\delta^2 + (m^2 + 6m - 14)\delta^3)$. The new algorithm is faster for the entire range of interest, i.e. $t \geq \delta \geq 1$ and $m \geq 2$. As a particular case, let $\delta = t$

and m =3; the IHU algorithm requires $82t^3$ operations, Yen's modification requires $128t^3$ operations, and Floyd's algorithm requires $250t^3$ operations.

3.4 Decomposing the Network for the IHU Algorithm

Perhaps even more important than the numerical gains of the new algorithm are the insights it provides into optimal decomposition of a network. Assume that Floyd's method is used for shortest route computations on submatrices, and that pseudo-multiplications are done by the straightforward technique. It follows that for a given decomposition, if a further decomposition exists by partitioning of existing sets, then the computation time of the further decomposition is less than that of the given decomposition. This "more the better" fact suggests a heuristically good decomposition technique which can be performed by the computer or quickly guessed at by eye. If the decomposition is to be done automatically by the computer, however, it should probably be limited to those cases where many shortest route computations for the same topology will be performed, as in column-generating linear programs. An algorithm for finding a good network decomposition for the IHU algorithm is:

(a) find two nodes, j and k, such that the minimal number of arcs, d, connecting them is maximal over all pairs of nodes; i.e. find the diameter of the network and an associated pair of nodes;

(b) construct d+1 sets by letting $S_1 = \{j\}$ and

$$S_{i+1} = \{m \mid m \in \{\Omega - \Omega_i\} \text{ and } d_{rm} < \infty \text{ for some } r \in S_i\}.$$

This procedure was used to generate the IHU decomposition sets for the examples of the next section, and the reader may want to look at the figures associated with that section at this point.

3.5 Some Examples Using the IHU and NXN Algorithms

In this section, several examples are presented which provide insight into the classes of networks for which the NXN and IHU algorithms can substantially reduce shortest route computation time. One can probably deduce from these examples that the NXN algorithm will perform a shortest route computation more rapidly on a "typical" network. However, the fourth example suggests that there is a small and special class of networks for which the IHU algorithm is more efficient. Typically, such networks may be decomposed in such a manner as to be a variation on the following theme: $|S_i|$ for i odd is large compared to $|S_i|$ for i even, and if $j \in S_i$ and $k \in S_i$ then j and k are very likely to have direct arcs to one another.

The first example is an old version of the ARPA net which is shown in Fig. 3.5.1. In that figure, the NXN decomposition is defined by the numbering of the nodes, and the IHU decomposition is defined by the partitioning of the nodes with broken lines. This network lends itself to NXN decomposition due to the large number of nodes which have arcs directly to only two other nodes--a fact which keeps the cardinality of connection sets very low.

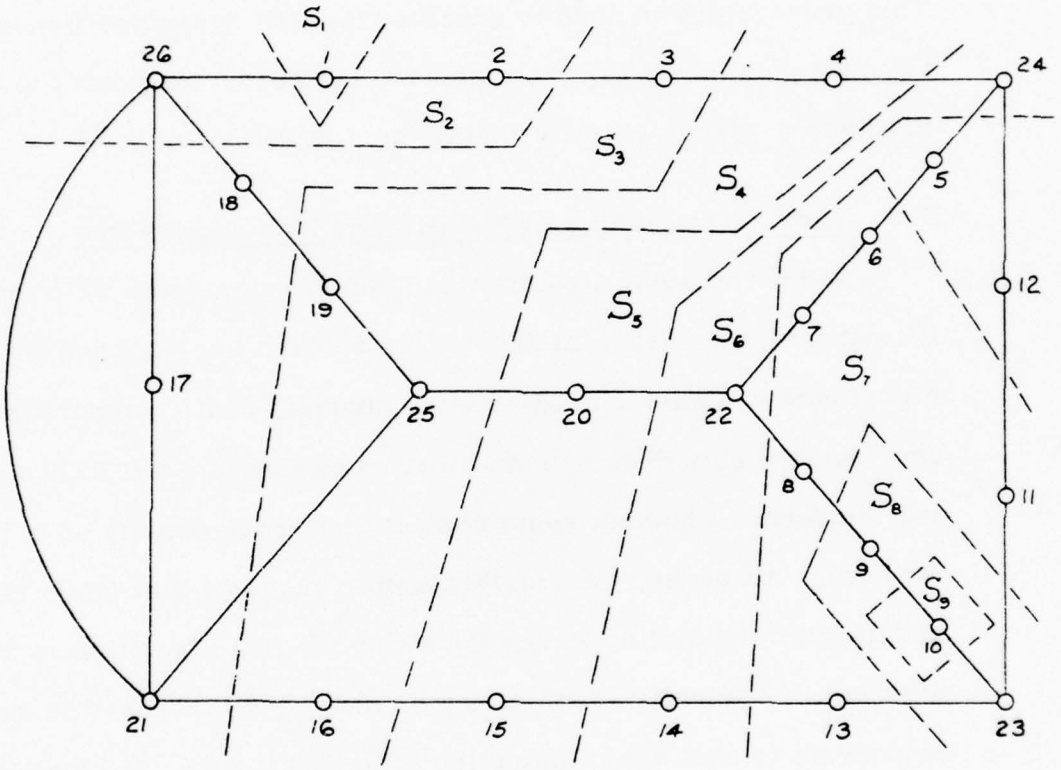


Fig. 3.5.1 The topology of ARPA network (at one stage of its evolution) with decomposition information.

The second example is the 47-node symmetric network shown in Fig. 3.5.2. This network is not "locally connected" to a very large degree, but still the NXN algorithm is (perhaps surprisingly) efficient.

The third example is the 64-node network displayed in Fig. 3.5.3. The density of arcs is perhaps greater here than in the other examples, but a large degree of local connectivity promotes the efficiency of the NXN algorithm.

The fourth example is the 36-node network of Fig. 3.5.4. This is typical of the type of network for which one would expect the IHU algorithm to be more efficient than the NXN algorithm.

Efficiency is measured with Yen's implementation of Dijkstra's algorithm as the standard. Theoretical efficiency refers to the relative savings in the number of operations required to perform a shortest route computation. The computation times for the IBM 370-168 to execute the Fortran program of the various algorithms were noted, and relative savings are referred to as the measured efficiency. The comparisons of the various algorithms in performing shortest route calculations on the three sample networks are summarized in Table 3.5.1. The Fortran programs were compiled by the IBM G1 compiler; and each algorithm not only computed the shortest distance matrix, but also computed a routing matrix which specified the first arc from each node on the shortest route to any other node.

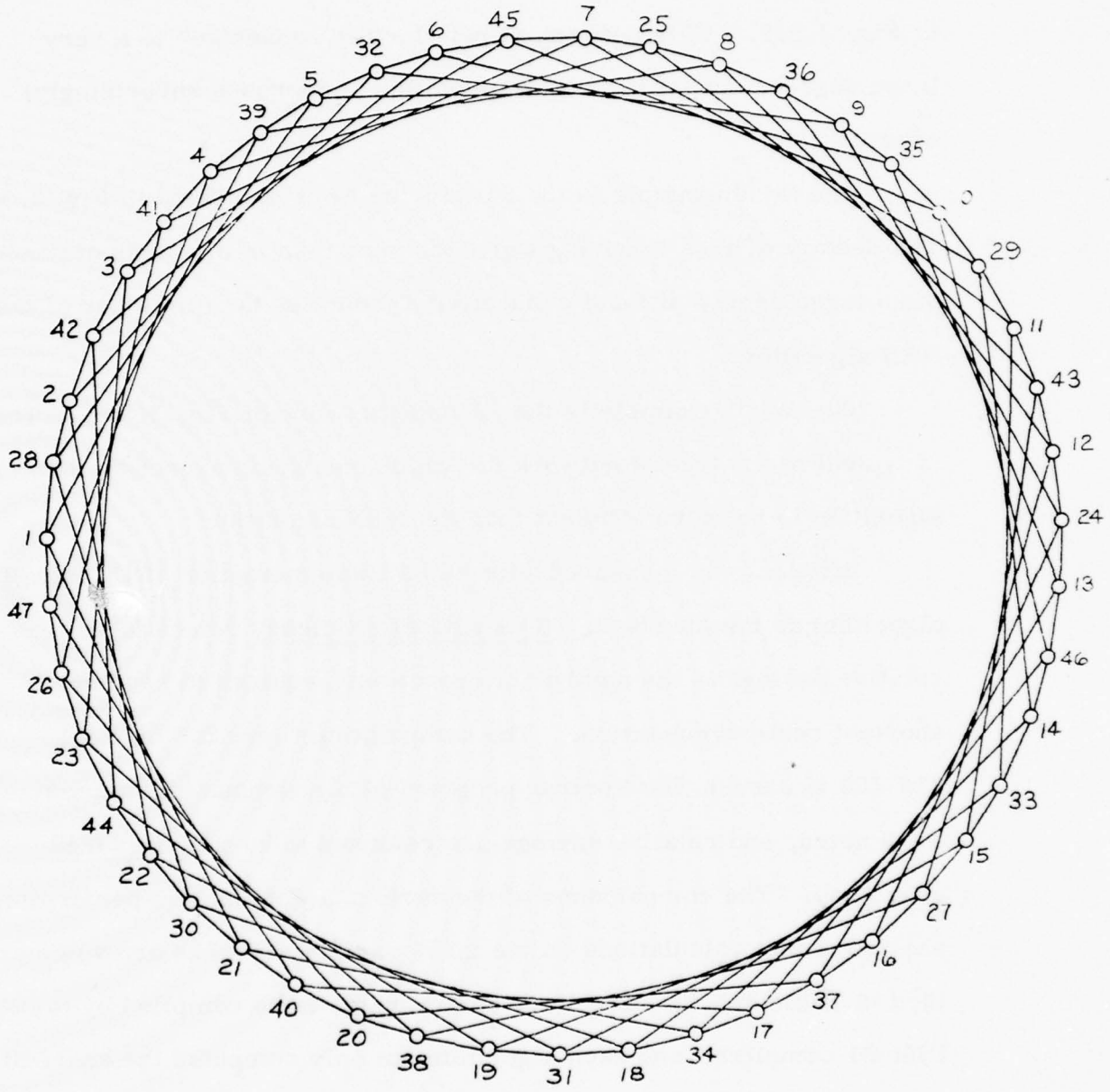


Fig. 3.5.2 A 47 node symmetric network (nodes are connected to first and seventh nearest neighbors by arcs). NXN decomposition is indicated by node labeling. IHU decomposition sets: $S_1 = \{1\}$, $S_2 = \{21, 47, 28, 39\}$, $S_3 = \{34, 40, 30, 26, 2, 4, 5, 8\}$, $S_4 = \{14, 17, 18, 29, 22, 23, 42, 41, 32, 25, 36, 43\}$, $S_5 = \{46, 33, 15, 37, 31, 38, 44, 3, 6, 7, 9, 29, 11, 12\}$, and $S_6 = \{13, 27, 16, 19, 45, 35, 10, 24\}$.

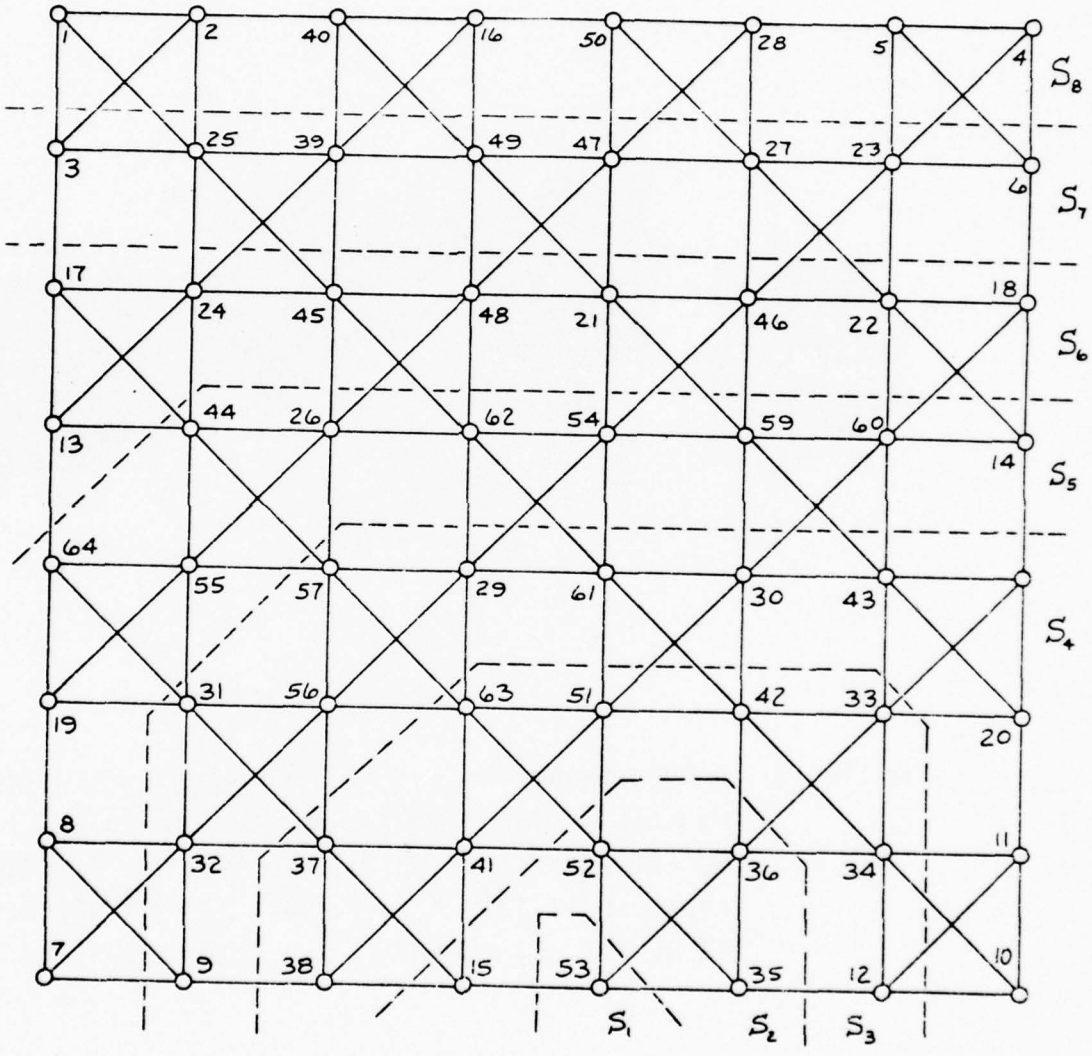


Fig. 3.5.3 A 64 node network with decomposition information.

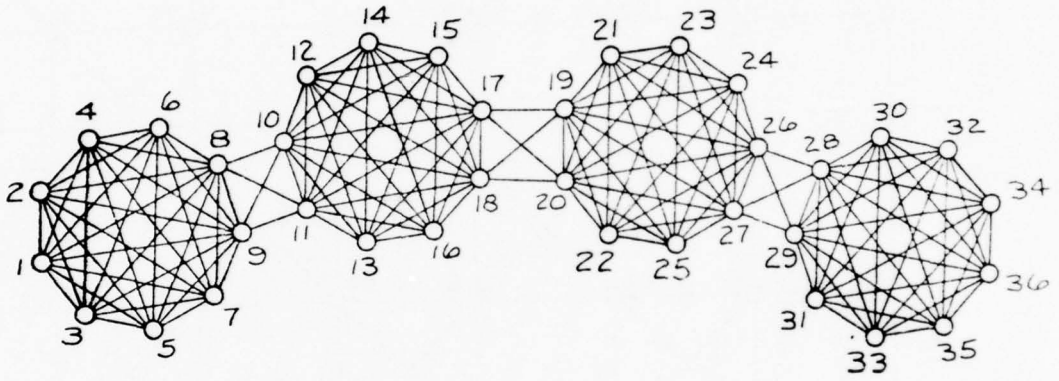


Fig. 3.5.4 A 36 node network. NXN decomposition is indicated by node labeling. IHU decomposition sets: $S_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $S_2 = \{10, 11\}$, $S_3 = \{12, 13, 14, 15, 16, 17, 18\}$, $S_4 = \{19, 20\}$, $S_5 = \{21, 22, 23, 24, 25, 26, 27\}$, $S_6 = \{28, 29\}$, $S_7 = \{30, 31, 32, 33, 34, 35, 36\}$.

		ARPA network of Fig. 3.5.1	47-node network of Fig. 3.5.2	64-node network of Fig. 3.5.3	36-node network of Fig. 3.5.4
Dijkstra's shortest route algorithm	Number of operations	27040	154630	393216	69984
	Theoretical efficiency	1.00	1.00	1.00	1.00
	Computation time (seconds)	0.090	0.450	1.115	0.210
	Measured efficiency	1.00	1.00	1.00	1.00
IHU algorithm	Number of operations	6948	83880	124722	11360
	Theoretical efficiency	3.89	1.84	3.15	6.16
	Computation time (seconds)	0.020	0.195	0.290	0.040
	Measured efficiency	4.50	2.31	3.84	5.25
NXN algorithm	Number of operations	2828	28608	42416	13316
	Theoretical efficiency	9.56	5.41	9.27	5.26
	Computation time (seconds)	0.015	0.115	0.165	0.050
	Measured efficiency	6.00	3.91	6.76	4.20

Table 3.5.1 Comparative performance of three different shortest route algorithms on the three sample networks.

3.6 Additional Notes on the Shortest Route Algorithms

For completeness, the method of bookkeeping for finding the shortest routes which was implemented in the algorithms encoded as a part of this thesis is presented. An N x N "next arc" matrix, $X = [x_{ij}]$, is established. where x_{ij} is the first arc on the current estimate of the shortest route from i to j. Initially, one sets $x_{ij} = m$ where $OR(m) = i$ and $DS(m) = j$ for every existing arc m. Whenever an operation $d_{ij} \leftarrow \min(d_{ij}, d_{ik} + d_{kj})$ is performed such that $d_{ik} + d_{kj}$ is the distinct minimum, then one makes the replacement $x_{ij} \leftarrow x_{ik}$. At the conclusion of a shortest distance algorithm, the shortest routes may be generated in an obvious fashion.

As a practical matter, it is noted that there are cases where a shortest distance algorithm succeeds in finding the shortest distances, but the bookkeeping for shortest routes fails. Failure is even more likely in a digital computer implementation than in an exact calculation (although both are possible). Sufficient conditions for success of the bookkeeping are as follows: (i) the network is connected, i.e. there exists a finite distance path between every ordered pair of nodes, and (ii) if d_{min} is the minimum distance assigned to any existing arc and d_{max} is the maximum distance assigned to any existing arc, then $d_{min} > 0$ and $d_{min} + d_{max}$ is distinct from d_{max} in the computer storage. Given these conditions as a hint, the interested reader should be able to construct an example where the bookkeeping fails although a shortest distance algorithm succeeds.

The use of the NXN and IHU algorithms should be limited to

networks of moderate size, i.e. under several hundred nodes. The basic reason for this limitation is that there are several algorithms that do not require storage of the entire D matrix and are consequently more efficient for very large sparse networks. Two of these, Dijkstra's algorithm (not Yen's version) and Bellman's algorithm, were compared by Golden [GOLD76]. From Golden's work, one can see that for large sparse networks the algorithms he compared are quite efficient; their performance, however, is not nearly so impressive for smaller networks. The failure of these algorithms which do not explicitly manipulate the D matrix to perform competitively on the smaller networks is owed to a large number of overhead operations associated with such algorithms.

CHAPTER IV

A LINEAR MULTICOMMODITY ROUTING PROBLEM

This chapter studies some methods for solving a linear multicommodity routing problem which yield a solution that has certain desirable properties in the context of computer networks. In essence, the solution to the linear routing problem places the minimum possible demand on the channels which form the "bottleneck" of the network. The linear problem is to find

$$\min_{f \in F} \left(\max_{1 \le i \le NA} f_i / C_i \right)$$

That is, one is minimizing the maximum utilization factor of any channel in the network over all possible routing strategies.

The channels associated with the binding capacity constraints in the solution to the linear program form the bottleneck of the network. Binding constraints, in this chapter, are those which are necessarily tight or active at the solution of the problem. They are "binding" in the sense that a decrease of the capacity of any associated channel implies an increase in the value of the objective. There are no flows through the bottleneck in the solution which even in part could avoid the bottleneck, and it is in this sense that the demand on the bottleneck is minimized.

From a slightly different point of view, this solution solves a throughput problem for the network. The inverse of the value of this linear objective function at the solution is the maximum scale factor

on the entire demand matrix such that a feasible flow (one which does not exceed any channel capacity) exists.

4.1 Decomposition Techniques for Solution to the Linear Routing Problem

The outlook of this chapter will be from the viewpoint of the Dantzig-Wolfe decomposition technique [DANT63]. The same approach could have been taken as in Chapter II, i.e. viewing the techniques as column generating algorithms. The problem which is to be solved,

$$\min_{f \in F} \left(\max_{1 \le i \le NA} \frac{f_i}{C_i} \right) \tag{4.1.1}$$

may be restated as:

minimize β_1 such that

$$\begin{aligned} -\beta_1 C + If^1 + If^2 + If^3 + \cdots + If^N &\le 0 \\ Ef^1 &= b^1 \\ Ef^2 &= b^2 \\ Ef^3 &= b^3 \\ \cdots & \\ Ef^N &= b^N \end{aligned} \tag{4.1.2}$$

$$f^k \ge 0 \text{ for } k \in \{1, 2, \cdots, N\}$$

where $C = [C_1, C_2, \cdots, C_{NA}]'$, β_1 is a real scalar, I is the NA x NA identity matrix, and the remainder of the notation is consistent with 1.1.2.

According to the Dantzig-Wolfe decomposition principle, one

may formulate the master problem of 4.1.2 as:

minimize β_1 such that

$$-\beta_1 C + \sum_{j=1}^{t_1} \lambda_{j1}\varphi^1_{(j)} + \sum_{j=1}^{t_2} \lambda_{j2}\varphi^2_{(j)} + \cdots + \sum_{j=1}^{t_N} \lambda_{jN}\varphi^N_{(j)} \leq 0$$

$$\sum_{j=1}^{t_1} \lambda_{j1} = 1$$

$$\sum_{j=1}^{t_2} \lambda_{j2} = 1$$

$$\cdots$$

$$\sum_{j=1}^{t_N} \lambda_{jN} = 1$$

$$\lambda_{jk} \geq 0 \text{ for every } j \in \{1, 2, \cdots, t_k\} \text{ and } k \in \{1, 2, \cdots, N\}$$

(4.1.3)

where the $\varphi^k_{(j)}$ are solutions to the various subproblems which will be defined momentarily, and t_k is the number of trees associated with the destination node k. The above format should be compared with that of 2.2.1. Upon solution to the restricted master, 4.1.3, there exists a set of dual variables, $\begin{bmatrix}\pi \\ \sigma\end{bmatrix} = [\pi_1, \pi_2, \cdots, \pi_{NA}, \sigma_1, \sigma_2, \cdots, \sigma_N]'$ associated with the constraints of the master; π_i is the dual variable associated with the ith arc, and σ_k is the dual variable associated with the constraint $\sum_{j=1}^{t_k} \lambda_{jk} = 1$.

There are N subproblems which are solved after each master, each having the form,

$$\text{minimize } -\pi' f^k \text{ such that}$$
$$Ef^k = b^k, \quad f^k \geq 0 \tag{4.1.4}$$

and the solution is denoted $\varphi^k_{(t_k+1)}$. Each $\varphi^k_{(t_k+1)}$ is a column which is added to the current master if its cost relative to the current solution is negative, i.e. if $(-\pi' \varphi^k_{(t_k+1)} - \sigma_k) < 0$. In the case that the column is added, $t_k \leftarrow t_k + 1$ in the master problem. It is noted that 4.1.4 is the same problem as 1.2.2, so that the method of solution is via shortest route algorithms. Consequently, the solutions to 4.1.4 are trees. It is known from linear programming that the dual variables associated with the arcs are nonpositive, assuring the success of shortest route algorithms for solving the subproblem [DANT63]. If all of the N trees generated by the subproblems have nonnegative relative cost, then the solution to 4.1.2 was established by the previous master problem and the algorithm terminates. Otherwise the master problem is resolved, and the above procedure is repeated.

The number of constraints in 4.1.3 is NA+N, which is also the maximal number of trees in a solution. $\hat{\beta}_1$, the value of β_1 at the solution of 4.1.3, is the minimal scale factor for the capacities such that there exists a feasible routing strategy, i.e. if $\delta < \hat{\beta}_1$ then there does not exist an $f \epsilon F$ such that $f_i \leq \delta C_i$ for every $i \epsilon \{1, 2, \cdots, NA\}$.

The above algorithm is denoted the LTR algorithm (linear routing algorithm based on a tree decomposition). Alternatively, 4.1.2 could have been expressed as:

$$\begin{aligned} &\text{minimize } \beta_1 \text{ such that} \\ &-\beta_1 C + If \leq 0 \\ &\qquad Mf \geq b \\ &\qquad f \geq 0 \end{aligned} \tag{4.1.5}$$

where the notation is consistent with 1.1.1. This problem may be solved via the Dantzig-Wolfe decomposition principle, and the procedure is similar to that of the EF algorithm which was presented in Section 2.1. The procedure is called the LEF algorithm, and the master problem is:

$$\begin{aligned} &\text{minimize } \beta_1 \text{ such that} \\ &-\beta_1 C + \sum_{i=1}^{j} \lambda_i \varphi_{(i)} \leq 0 \\ &\sum_{i=1}^{j} \lambda_i = 1 \\ &\lambda_i \geq 0 \end{aligned} \tag{4.1.6}$$

with a subproblem analogous to that of 4.1.4 and a similar criterion for terminating the algorithm. The solution to 4.1.2 has no more than NA+1 extremal flows in the solution.

Finally, 4.1.2 may be formulated as an N^2-commodity flow problem using the notation of 1.1.3. Similarly, a decomposition scheme for solving the problem based on chains can be derived, and the resultant algorithm is called the LCH algorithm. Accordingly, the solution to 4.1.2 would have no more than NA+N(N-1) chains in the solution. Note that upon finding the solution to 4.1.1, the associated

routing strategy is derived in the same manner as presented in Section 2.1 with varying amounts of difficulty according to the decomposition scheme and its format for the solution.

4.2 Higher Order Criteria for the Linear Program Routing Strategy

This section discusses higher order criteria for the solution to 4.1.1. There are many members of F which solve 4.1.1, and the purpose of the criteria is to define the "best" among the many solutions. For the sake of compact notation, the criteria are exemplified in terms of the LEF algorithm, with the extensions to the LTR algorithm and the LCH algorithm being sufficiently straightforward as to be omitted. Assume that 4.1.1 has been solved by the LEF algorithm and its solution is $\hat{\beta}_1$.

Having solved 4.1.6, some of the tight capacity constraints are associated with a dual variable which is negative. It is an immediate consequence of duality [DANT63, pp. 134-40] that such a constraint has no slack in any solution of 4.1.6; hence, any reduction of the capacity of the associated channel would result in an increase in the objective function. Of course, every channel in the bottleneck (i.e. every channel corresponding to a binding constraint) is not necessarily indicated by a negative dual variable; however, provisions for such omissions are included in the following discussion.

Define S_1 (called the first saturation set) as follows: $i \in S_1$ if $\pi_i < 0$ in the solution of 4.1.6. Also, define $\delta_{i,S_1} = 1$ if $i \in S_1$, and $\delta_{i,S_1} = 0$ if $i \notin S_1$. Then the secondary criterion for the solution to the

linear program is defined to be that the solution to 4.1.1 also solves:

$$\text{minimize } \beta_2 \text{ such that}$$

$$-\beta_2 \bar{C}^1 + If \leq \hat{\beta}_1 C^1$$

$$Mf \geq b$$

$$f \geq 0$$

where $C^1 = [c_{1,1}, c_{1,2}, \cdots, c_{1,NA}]'$ such that $c_{1,i} = C_i \delta_{i,S_1}$, and where $\bar{C}^1 = [\bar{c}_{1,1}, \bar{c}_{1,2}, \cdots, \bar{c}_{1,NA}]'$ such that $\bar{c}_{1,i} = C_i(1-\delta_{i,S_1})$. Note that $C^1 + \bar{C}^1 = C$. The physical interpretation of the problem is to minimize the maximal utilization factor of any channel other than those in S_1 given that the utilization factor of the channels of S_1 are fixed at $\hat{\beta}_1$. If the value of the objective of the secondary problem is equal to $\hat{\beta}_1$, then the entire bottleneck is not indicated by S_1. In this case the channels associated with negative dual variables in the solution to the secondary problem are also part of the bottleneck. However, computational experience indicates that if the elements of the demand matrix are arbitrarily chosen and few are zero, then it is fairly likely that S_1 indicates the entire bottleneck.

The jth order criterion is a straightforward generalization of the secondary criterion. The reader will be spared the additional notational complexities, although the interpretation is as follows. For every criterion, there are constraints in the solution of the associated problem which are for the first time assigned negative dual variables. The utilization factors of the corresponding channels are

then fixed at their value in the solution. The problem associated with the next criterion is to minimize the maximal utilization factor over those channels which have not had their utilization factors fixed by a previous criterion.

4.3 Examples Using the LEF, LTR and LCH Algorithms

Several examples which provide insight into the behavior of the linear programming algorithms are presented in this section. A linear programming system entitled SEXOP (Subroutines for EXperimental OPtimization), which is currently maintained at MIT by Prof. Roy Marsten [MARS74], was employed for these examples in the master problem. Some limitations on the size of the examples were imposed by the capabilities of SEXOP. The principal constraints were that no network with more than 99 directed channels could be handled, and the LCH algorithm could not be used on networks with more than 14 nodes.

The four example networks are shown in Figs. 4.3.1, 4.3.2, 4.3.3, and 4.3.4. The requirement matrix for each of the problems was pseudo-randomly generated according to the rules of Appendix C. The arcs are assumed to be duplex and have unit capacity. The arcs marked by arrows in the figures denote the channels associated with the binding constraints in the solution of the specified problem. The algorithms successively refine the estimate of $\hat{\beta}_1$, i.e. β_1, until a solution is obtained, and graphs of β_1 versus computation time are presented in Figs. 4.3.5, 4.3.6, 4.3.7, and 4.3.8.

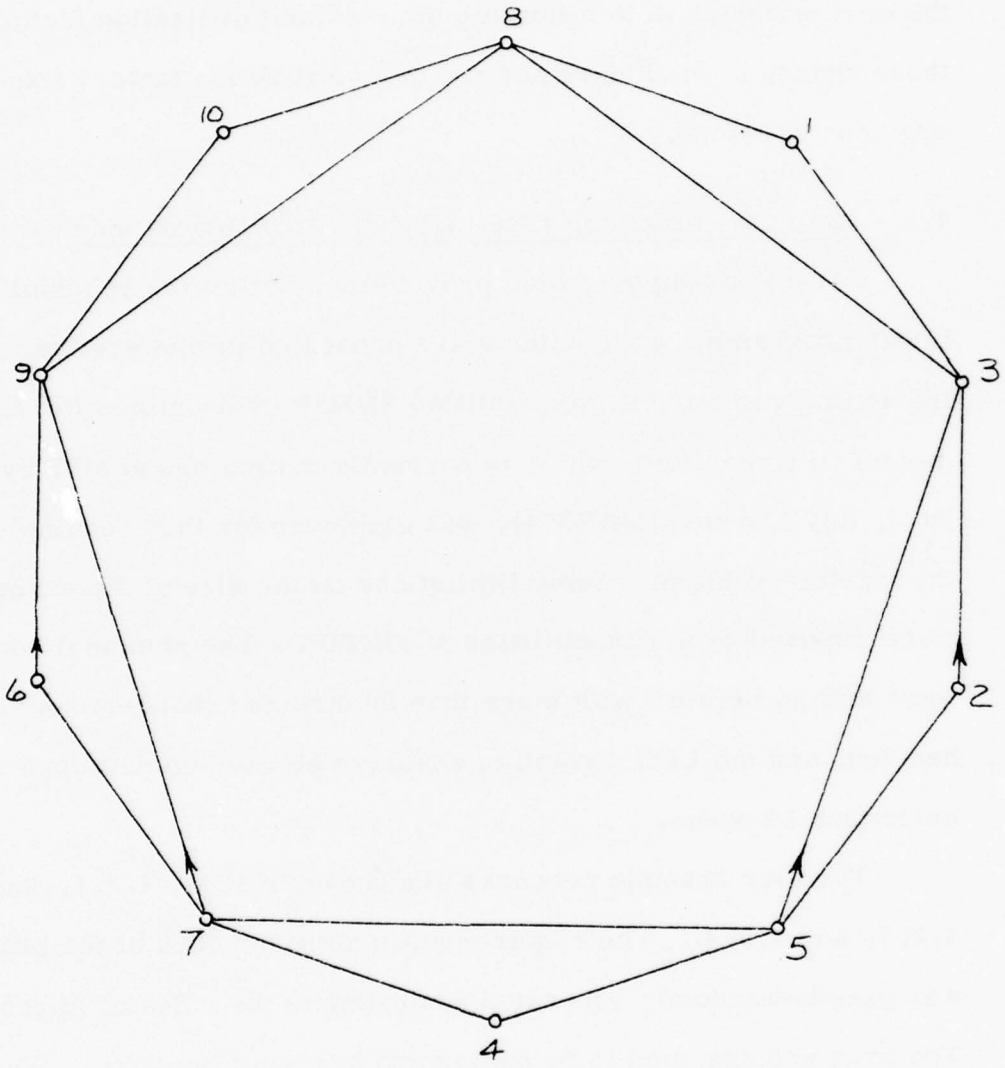


Fig. 4.3.1 A ten node network with thirty directed arcs.

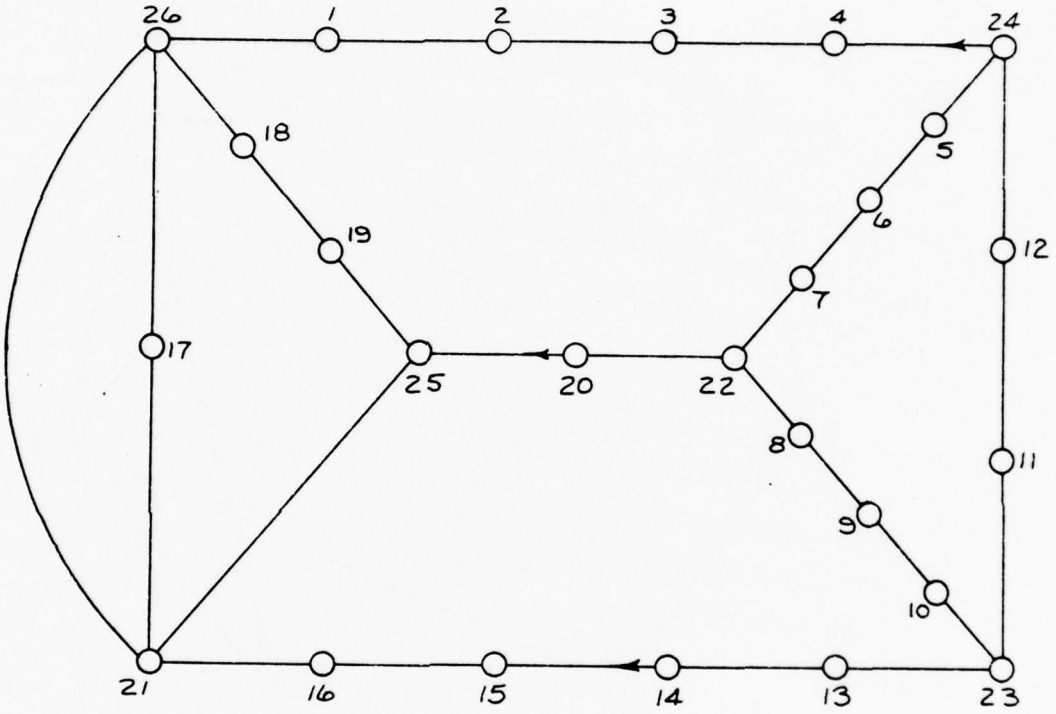


Fig. 4.3.2 A twenty-six node, sixty directed arc network.

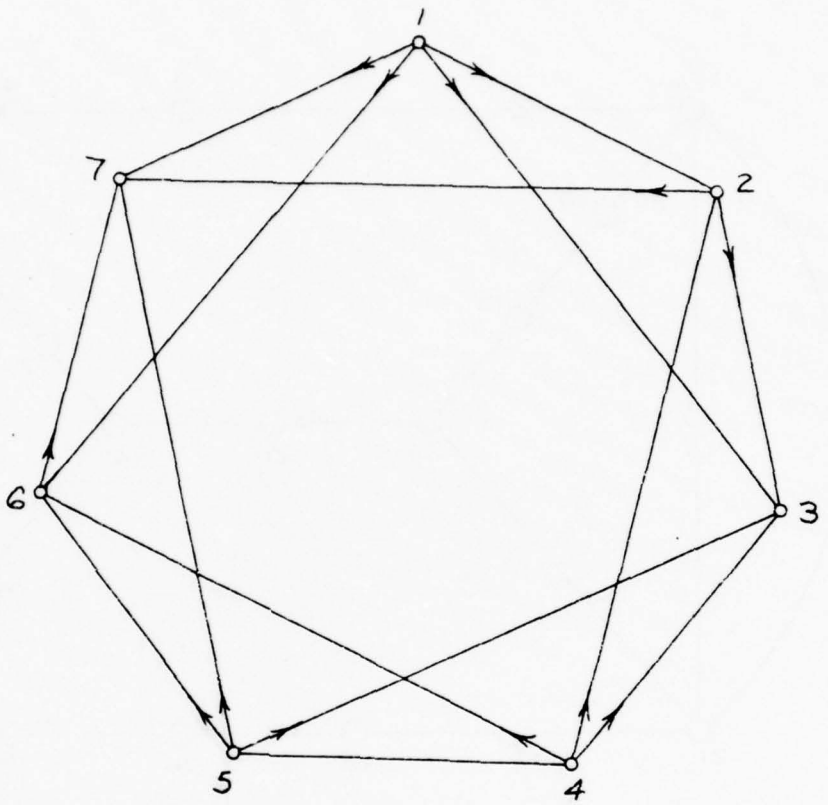


Fig. 4.3.3 A seven node symmetric network with twenty-eight directed arcs.

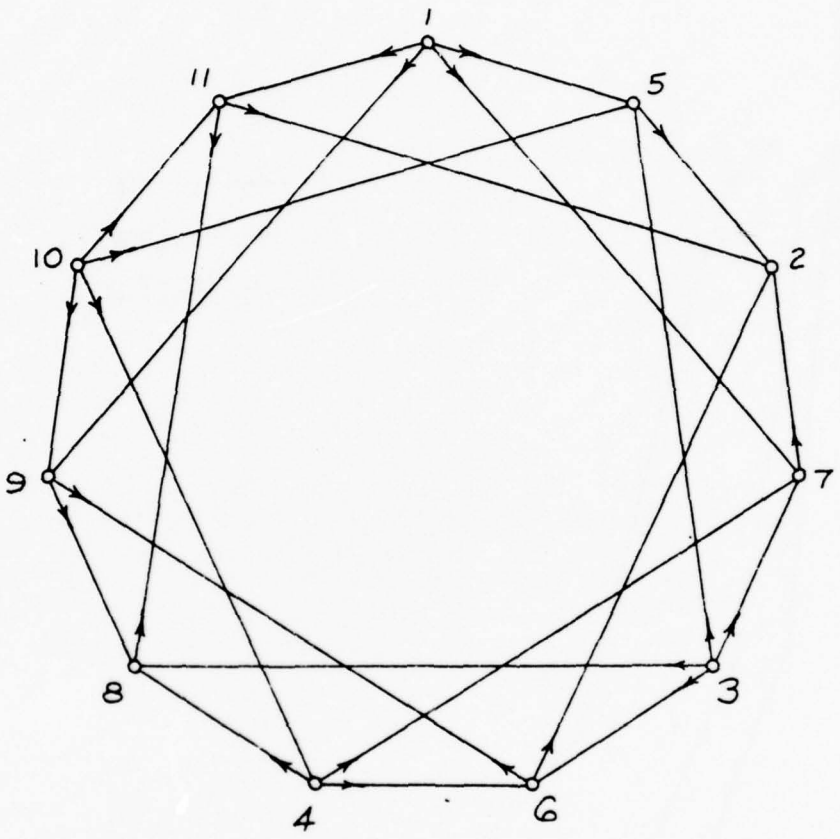


Fig. 4.3.4 An eleven node symmetric network with forty-four directed arcs.

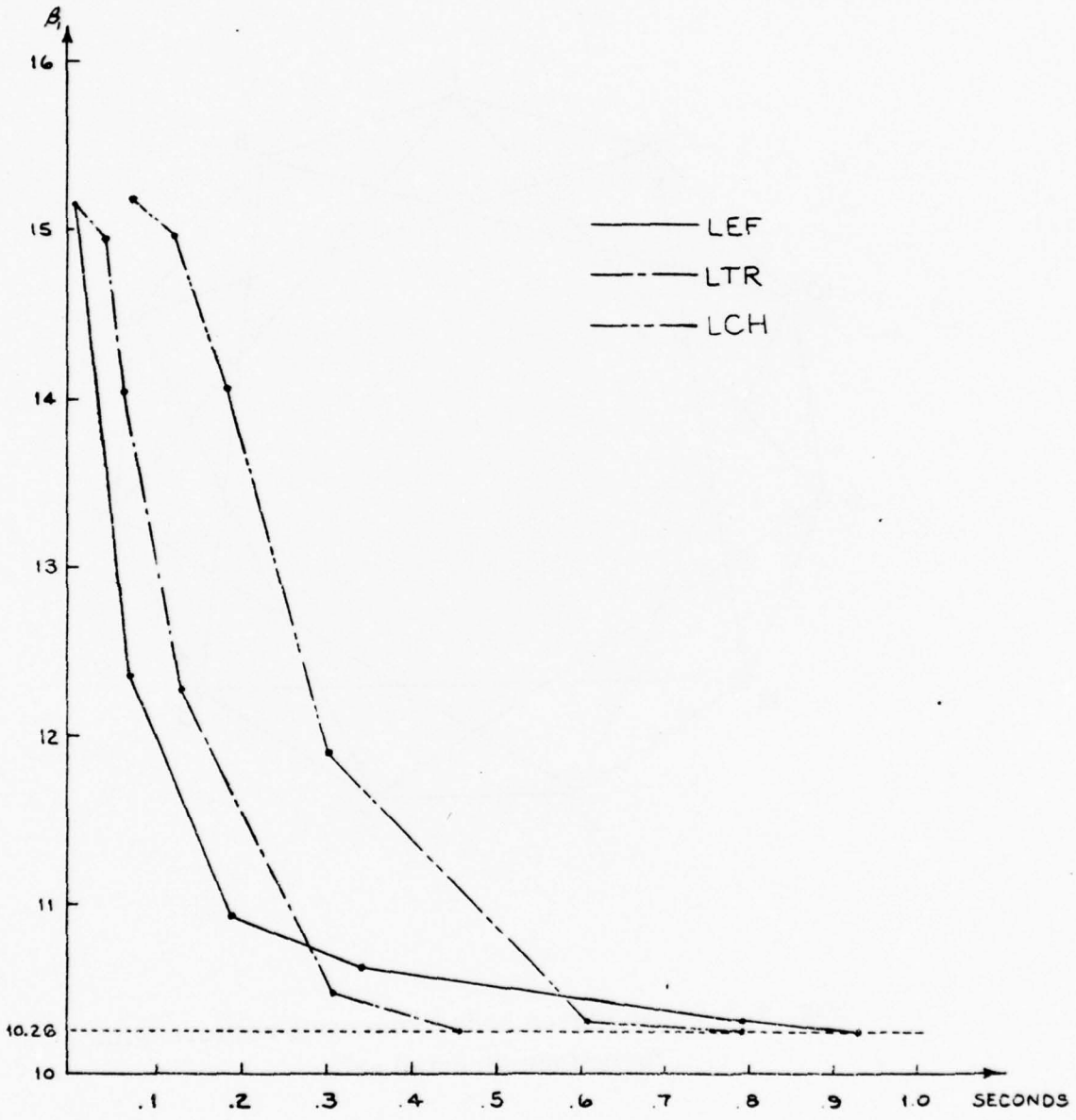


Fig. 4.3.5 Experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.1 for the LEF, LTR and LCH algorithms.

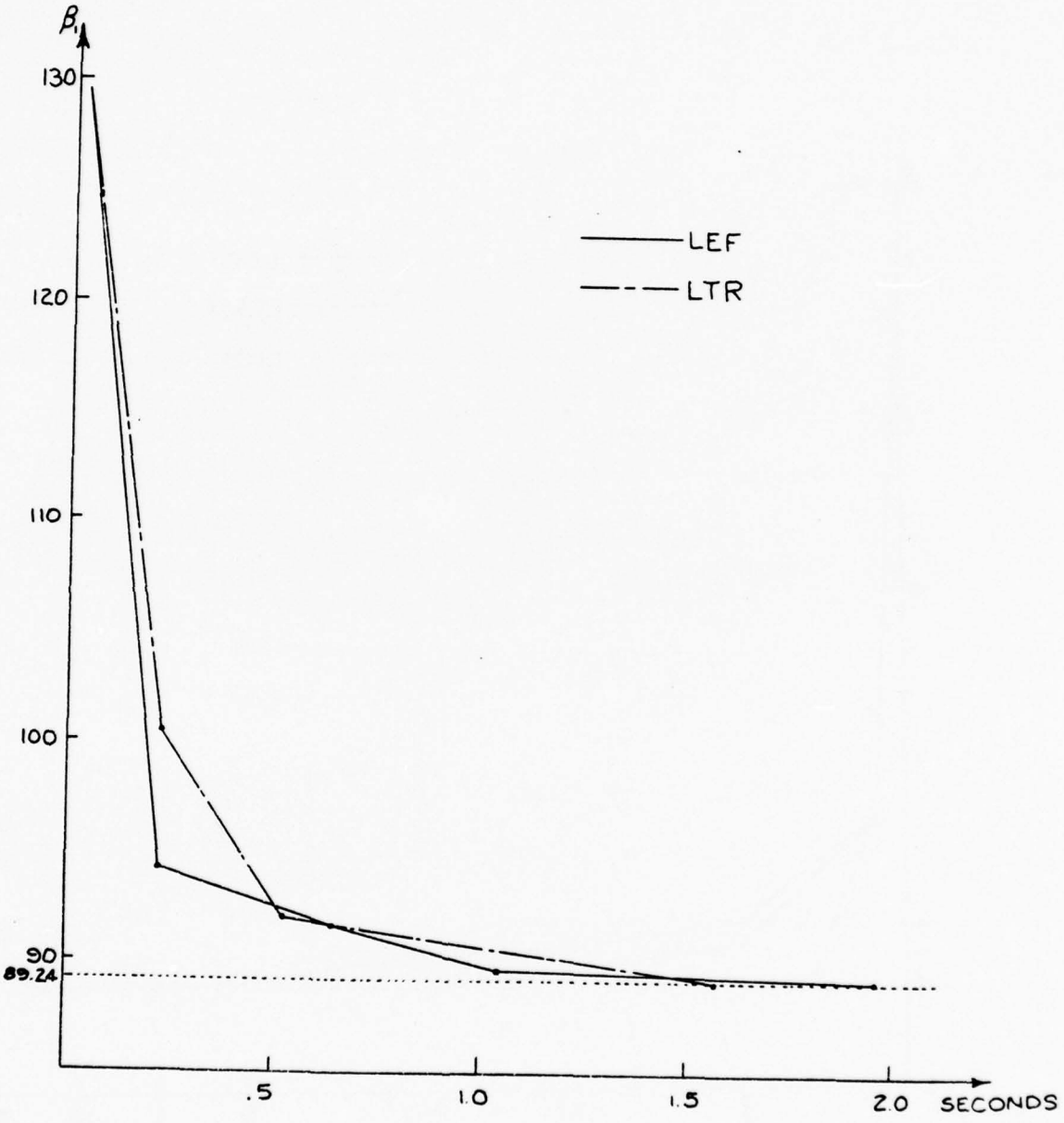


Fig. 4.3.6 Experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.2 for the LEF and LTH algorithms (the LCH algorithm was not run for this problem).

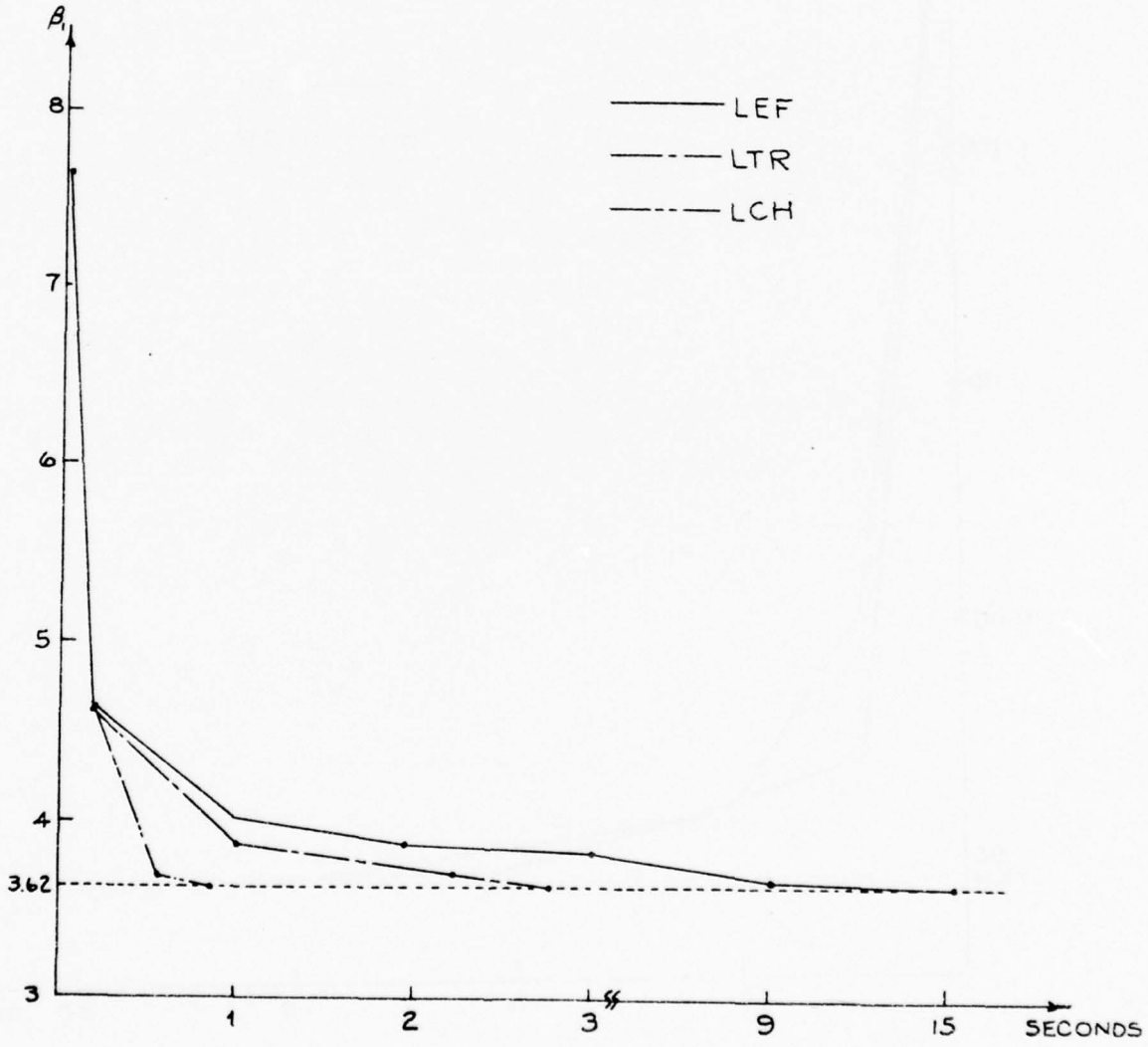


Fig. 4.3.7 Experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.3 for the LEF, LTR and LCH algorithms (note the change of horizontal scale at 3 seconds).

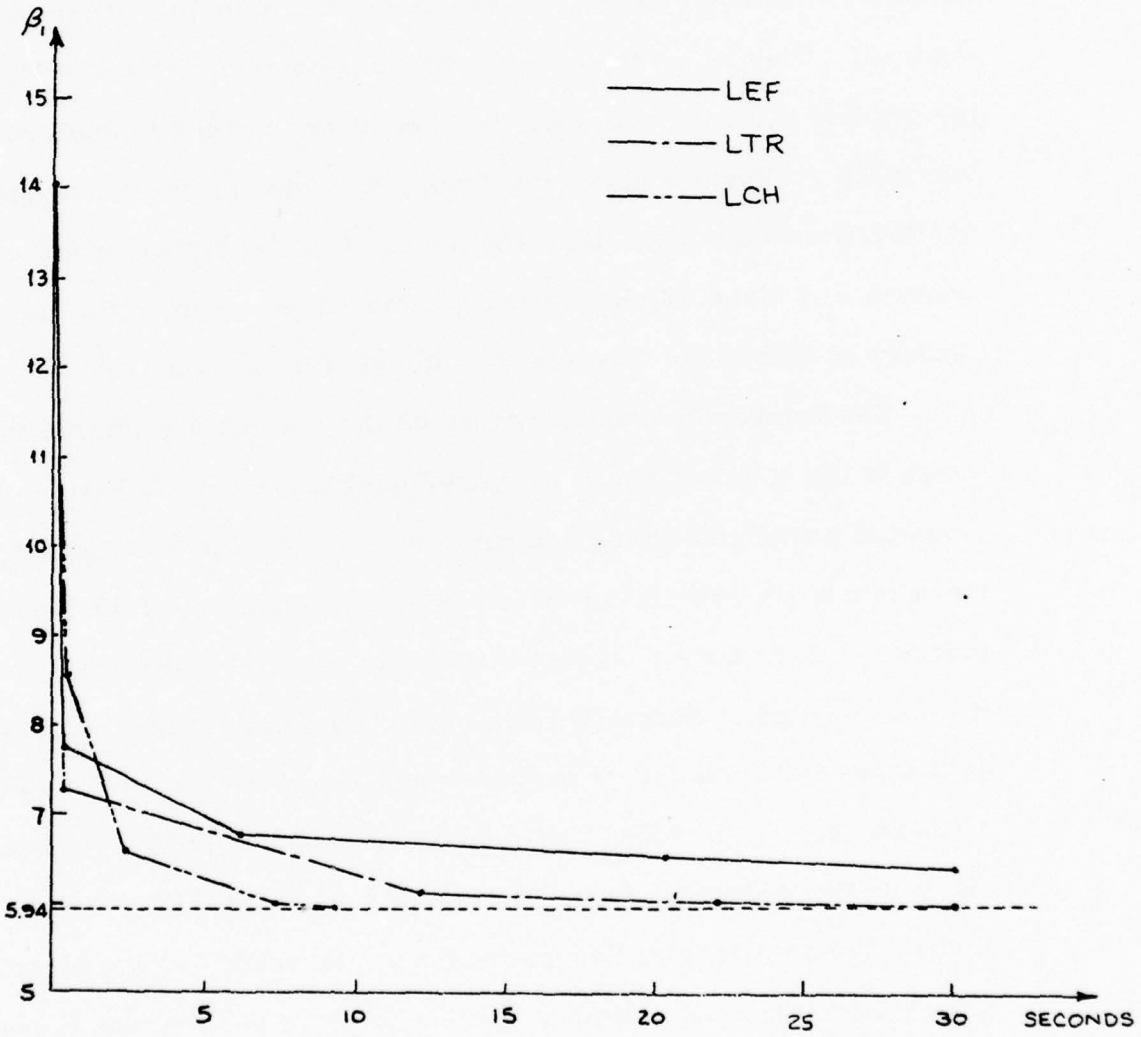


Fig. 4.3.8 Experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.4 for the LEF, LTR and LCH algorithms. The LEF algorithm still had not reached the solution at 90 seconds when its value of β_1 was 6.00.

It is noted that in the case of unit capacity for every arc, the linear routing problem becomes: $\min_{f \in F} \max_{1 \le i \le NA} f_i$. Thinking of channels in the queueing sense, and the problem as minimizing the maximal utilization factor, then values of the objective under one are desired. This is not the case for these problems, as is displayed in the graphs, but one may scale the capacities in order to achieve feasibility. Furthermore, the graphs only display the convergence for the first criterion. Refining the solution for higher order criteria was found to consume large amounts of computation time relative to that of the first order criterion (see Appendix D).

The comparative speeds at which the various algorithms converge to the solution can be explained qualitatively as follows. The principal component in the computation time of a linear program is that associated with the construction and maintenance of the basis inverse. Each time a column is added to the basis, the basis inverse must be altered. Naturally, one would like to minimize the number of changes in the basis. Consider being very "far" from the solution. The addition of one chain to the basis will probably have a very modest effect on the objective. The addition of a tree would have more power, and an extremal flow even more. However, as the algorithm closes in towards the solution, the flexibility of the columns becomes the most important factor in the convergence, and the three types of columns would be desirable in the reverse order of their power during the initial stages of the algorithm.

The conclusion is that one would expect the LEF algorithm to

be the fastest starter and the LCH algorithm to be the fastest finisher. If the problem is very complicated in the sense that many constraints are binding in the solution, then the finish is far more important than the start and the LCH algorithm will be the clear winner. If the problem is not very complicated, then a faster starting algorithm may also be the first one to solve the problem (see Fig. 4.3.5).

The conclusions from the linear programming examples are fairly clear. The LEF algorithm (which is very much related to the method Hu suggests for solving the problem [HU69]) will rarely be the best technique for solving the problem. The LTR algorithm has a compact representation in the machine and has good speed characteristics relative to the LEF algorithm and should nearly always be preferred to the LEF algorithm. The only drawback of the LCH algorithm, which is fastest for the more difficult problems where speed is most important, is the very large number of commodities. The columns of the LEF and LTR algorithms have a uniform number of nonzero elements. A general purpose linear programming package like SEXOP handles the storage of the columns efficiently. However, the unpredictability of the number of nonzero elements of the columns of the LCH algorithm compounded by the failure to exploit the fact that all the nonzero elements in any column are equal, makes storage in the SEXOP package wasteful. A special purpose linear program would alleviate some of the storage burdens.

4.4 A Nonlinear Programming Approach to the Linear Routing Problem

This section opens with a presentation of an algorithm which is based on nonlinear optimization techniques, but which converges to the solution on the linear routing problem. As an aid, a lower bound on $\hat{\beta}_1$ is derived which may be used to make an intelligent decision on when to terminate the algorithm.

Prior to a formal statement of the algorithm, the intuition behind the algorithm is presented. The parameter σ is thought of as a scaling factor on the capacity vector of the network. An arbitrary $f \epsilon F$ is selected as the starting point for the algorithm. An initial σ is chosen such that f is feasible, i.e. $f < \sigma C$. A minimization is performed with an objective function which has a pole at each $f_i = \sigma C_i$. This tends to drive down the higher values of f_i / C_i at the expense of the others. Then σ is rechosen such that the slack in the problem is reduced. However, the new value of σ must not destroy the feasibility of the current flow vector (which is thought of as the current estimate of the solution to 4.1.1). The algorithm either stops according to some criterion or iterates by returning to the minimization step.

The algorithm. The following algorithm will be denoted $AL^*(\alpha, \epsilon, \gamma)$:

(1) Choose arbitrarily any $f^0 \epsilon F$. Define $\beta_1(f) = \max_{1 \le i \le NA} (f_i / C_i)$, and set $\sigma_1 = \gamma \beta_1(f^0)$ where γ has been chosen greater than 1. Let K be an iteration counter, and set K = 1.

(2) To within a predetermined tolerance, solve the following

problem (starting from the flow vector f^{K-1}):

$$\min_{f \in F} P(\sigma_K, f)$$

where

$$P(\sigma, f) = \sum_{j=1}^{NA} \frac{\sigma C_j}{\sigma C_j - f_j} = \sum_{j=1}^{NA} \frac{1}{1 - \rho_j/\sigma}$$

and where the utilization factor f_i / C_i is denoted by ρ_i. That is, find an f^K such that

$$P(\sigma_K, f^K) \leq (1 + \epsilon) \min_{f \in F} P(\sigma_K, f)$$

where ϵ has been chosen as an arbitrary positive constant. This tolerance may be guaranteed by the upper bound on the error given in Chapter II.

(3) Set $\sigma_{K+1} = (1-\alpha)\sigma_K + \alpha\beta_1(f^K)$, assuming α has been chosen such that $0 < \alpha < 1$. If $K = K_o$, where K_o is a prespecified number of iterations, cease the algorithm. Otherwise, perform $K \leftarrow K+1$, and go to step 2.

From Chapter II, one has a choice of several algorithms to use in step 2. Each performs the required step in a finite number of operations. If the EF algorithm is employed, the above algorithm is denoted $EF^*(\alpha, \epsilon, \gamma)$, and analogous notation is used for the other algorithms.

Because of the finite effort involved in step 2, it is now shown that $AL^*(\alpha, \epsilon, \gamma)$ converges to a δ tolerance solution of the linear routing problem in a finite number of operations; i.e. for any $\delta > 0$,

there exists a $K_o < \infty$ such that $\beta_1(f^{K_o}) \leq \delta + \hat{\beta}_1$. The proof is not difficult. Consider step 2. Let g be any solution of the linear routing problem. Then

$$\min_{f \in F} P(\sigma_K, f) \leq P(\sigma_K, g) \leq \frac{\sigma_K NA}{\sigma_K - \hat{\beta}_1}$$

since each element of g satisfies $g_i / C_i \leq \hat{\beta}_1$. This implies

$$P(\sigma_K, f^K) \leq \frac{NA(1+\epsilon)\sigma_K}{\sigma_K - \hat{\beta}_1}$$

Clearly, $\sigma_K / (\sigma_K - \beta_1(f^K)) \leq P(\sigma_K, f^K)$ since the former is one term of the summation which defines the latter, and it follows that

$$\frac{\sigma_K}{\sigma_K - \beta_1(f^K)} \leq \frac{NA(1+\epsilon)\sigma_K}{\sigma_K - \hat{\beta}_1}$$

Note that $\sigma_K > \beta_1(f^K) \geq \hat{\beta}_1 > 0$ (neglecting the trivial case $\hat{\beta}_1 = 0$) by construction of the algorithm and the definition of $\hat{\beta}_1$. Now using the equation in step 3, one finds that

$$\begin{aligned} \sigma_{K+1} &\leq \sigma_K - \alpha \left(\frac{1}{NA(1+\epsilon)} \right) (\sigma_K - \hat{\beta}_1) \\ &\leq \sigma_K - r(\sigma_K - \hat{\beta}_1) \end{aligned} \qquad (4.4.1)$$

where $1 > r = \alpha / (NA(1+\epsilon)) > 0$. This implies that σ_K converges to $\hat{\beta}_1$ at least geometrically, and the convergence of $AL^*(\alpha, \epsilon, \gamma)$ is proven.

A lower bound on $\hat{\beta}_1$.† Let S be any set of arcs in the network. S should be chosen as an estimate of the bottleneck associated with the linear routing problem to achieve a good lower bound on $\hat{\beta}_1$. Furthermore, define:

$$P_S(\sigma, f) = \sum_{j \in S} \frac{1}{1 - \rho_i/\sigma}$$

Using the same methodology employed in Section 2.1, one can find a lower bound for the value of $P_S(\sigma, g)$ for any $g \in F$. Therefore,

$$\epsilon_S(\sigma, f) \triangleq \langle \nabla P_S(\sigma, f),\ f - \varphi \rangle \geq P_S(\sigma, f) - P_S(\sigma, g)$$

where φ solves $\min_{h \in F} \langle \nabla P_S(\sigma, f), h \rangle$. In particular, let g be any solution to 4.1.1. Then it is clear that

$$P_S(\sigma, g) \leq |S| \frac{1}{1 - \hat{\beta}_1/\sigma}$$

since every component of g satisfies $f_i/C_i \leq \hat{\beta}_1$. Thus,

$$|S| \frac{1}{1 - \hat{\beta}_1/\sigma} \geq P_S(\sigma, g) \geq P_S(\sigma, f) - \epsilon_S(\sigma, f)$$

which yields a lower bound for $\hat{\beta}_1$, that is:

$$\hat{\beta}_1 \geq \sigma - \frac{|S|\sigma}{P_S(\sigma, f) - \epsilon_S(\sigma, f)} \triangleq \beta_L(S, \sigma, f)$$

Some trade-offs. Now suppose that f^K approximately solves $\min_{g \in F} P(\sigma_K, g)$ for a σ_K "very close" to $\hat{\beta}_1$ and an ϵ sufficiently small. Then S should be easy to identify, since channels which can have a

† For a proof of the convergence of the lower bound, see Appendix A, Thm 4.4.A.

utilization factor "far" from σ_K will tend to be driven down by the objective function. Also, $\epsilon_S(\sigma_K, f^K)$ should be small since the terms of $P_S(\sigma_K, f)$ are the dominating terms of $P(\sigma_K, f)$ in the neighborhood of f^K. Under these assumptions $\beta_L(S, \sigma_K, f^K)$ will not be far from $\hat{\beta}_1$ (as one can see by the substitutions $\epsilon_S(\sigma_K, f^K) \approx 0$ and $P_S(\sigma_K, f^K) \approx |S|/(1 - \hat{\beta}_1/\sigma_K)$ into the above expression for $\beta_L(S, \sigma_K, f^K)$).

Some trade-offs are now presented. One cannot make ϵ arbitrarily small because at some point the rate of convergence of the $AL^*(\alpha, \epsilon, \gamma)$ algorithm will suffer, and yet a fairly small ϵ is clearly necessary for the lower bound on $\hat{\beta}_1$ to be tight.

A "lower bound directed" algorithm would keep ϵ small. One might embellish the algorithm by heavily weighting those terms of $P(\sigma_K, g)$ corresponding to the channels which one estimates are most likely in the bottleneck. If one guesses right, the behavior of $P(\sigma_K, f)$ in the neighborhood of f^K will be more similar to that of $P_S(\sigma_K, f)$, and the control on the magnitude of $\epsilon_S(\sigma_K, f^K)$ will be enhanced.

Despite the foregoing possibility, a "convergence directed" algorithm was found to be more useful for the purposes of this thesis after experimentation with each type. In essence one opts to allow the lower bound to degrade in an attempt to speed up the algorithm. Towards this same goal, minimization of $P(\sigma, f)$ was replaced in the definition of $AL^*(\alpha, \epsilon, \gamma)$ by a minimization of

$$P^*(\sigma, f) = \sum_{i=1}^{NA} \ln\left(\frac{\sigma C_i}{\sigma C_i - f_i}\right)$$

The advantage of $P^*(\sigma, f)$ over $P(\sigma, f)$ lies in the form of the gradient

whose terms (i.e. $\partial P^*(\sigma, f)/\partial f_i = 1/(\sigma C_i - f_i)$) are first rather than second order poles. Due to the number of times which the gradient is calculated, the simplicity of the gradient translates into significant time reductions. The disadvantage is that the behavior of $P^*(\sigma_K, f)$ in the neighborhood of f^K does not resemble that of $P_S(\sigma_K, f)$ as well as the behavior of $P(\sigma_K, f)$ does. This tends to degrade $\epsilon_S(\sigma_K, f^K)$ and consequently $\beta_L(S, \sigma_K, f^K)$.

The deterioration of the lower bound is in fact so severe that its utility is virtually destroyed. One is therefore motivated to examine how $\beta_L(S, \sigma, f^K)$ behaves as a function of σ; specifically, no longer is the σ of $\beta_L(S, \sigma, f^K)$ taken to be synonymous with the σ_K of $P^*(\sigma_K, f)$.

As σ grows larger than σ_K, the behavior of $P_S(\sigma, f)$ in the neighborhood of f^K becomes even less similar to that of $P^*(\sigma_K, f)$, which is discouraging. On the other hand, as σ grows and $P_S(\sigma, f)$ tends to behave more linearly about f^K, $\epsilon_S(\sigma, f^K)$ becomes a less liberal estimator of the maximal error in $P_S(\sigma, f)$, which favors the growth of σ. Consider the following:

$$\lim_{\sigma \to \infty} \beta_L(S, \sigma, f) = \lim_{\sigma \to \infty} \left(\sigma - \frac{|S|\sigma}{P_S(\sigma, f) - \epsilon_S(\sigma, f)} \right)$$

$$= \lim_{\sigma \to \infty} \left(\sigma - \frac{\sum_{i \in S} \sigma}{\sum_{i \in S} \frac{1}{1 - \rho_i/\sigma} - \max_{\varphi \in F} \sum_{i \in S} \frac{(f_i - \varphi_i)/\sigma C_i}{(1 - \rho_i/\sigma)^2}} \right)$$

$$= \lim_{\sigma \to \infty} \left(\sigma - \frac{\sum_{i \in S} \sigma}{\min_{\varphi \in F} \sum_{i \in S} \frac{1 - 2\rho_i/\sigma + \varphi_i/\sigma C_i}{(1 - \rho_i/\sigma)^2}} \right)$$

$$= \lim_{\sigma \to \infty} \frac{\min_{\varphi \in F} \left(\sum_{i \in S} \frac{\sigma - 2\rho_i + \varphi_i/C_i}{(1 - \rho_i/\sigma)^2} \right) - \sum_{i \in S} \sigma}{\min_{\varphi \in F} \sum_{i \in S} \frac{1 - 2\rho_i/\sigma + \varphi_i/\sigma C_i}{(1 - \rho_i/\sigma)^2}}$$

$$= \lim_{\sigma \to \infty} \frac{\min_{\varphi \in F} \sum_{i \in S} \frac{\varphi_i/C_i - \rho_i^2/\sigma}{(1 - \rho_i/\sigma)^2}}{\min_{\varphi \in F} \sum_{i \in S} \frac{1 - 2\rho_i/\sigma + \varphi_i/\sigma C_i}{(1 - \rho_i/\sigma)^2}}$$

$$= \frac{\min_{\varphi \in F} \sum_{i \in S} (\varphi_i/C_i)}{|S|}$$

In order to gain insight into the utility of $\beta_L(S, \infty, f)$, the problem can be simplified. Let $C_i = 1$ for every $i \in \{1, 2, \cdots, NA\}$, so that

$$\beta_L(S, \infty, f) = (1/|S|) \min_{\varphi \in F} \sum_{i \in S} \varphi_i$$

This can be evaluated via a shortest route algorithm with a distance of 1 assigned to each arc which is estimated to be in the bottleneck and a distance of 0 on the remaining channels.

The condition for which $\beta_L(S, \infty, f)$ is in fact $\hat{\beta}_1$ is derived as follows.

Let $g \in F$ be a solution of 4.1.1, and assume that g is known as a convex combination of chains, i.e. $g = \sum_{k=1}^{N} \sum_{\substack{m=1 \\ m \neq k}}^{N} \sum_{j=1}^{c_{mk}} \lambda_{jmk} \varphi_{(j)}^{mk}$. For S chosen as the bottleneck, it follows that:

$$\hat{\beta}_1 = (1/|S|) \sum_{i \in S} g_i$$

$$= (1/|S|) \sum_{k=1}^{N} \sum_{\substack{m=1 \\ m \neq k}}^{N} \sum_{j=1}^{c_{mk}} \lambda_{jmk} r_{mk} \ell_{(j)}^{mk}$$

where $\ell_{(j)}^{mk}$ = the number of bottleneck channels used by chain $\varphi_{(j)}^{mk}$. Thus:

$$\hat{\beta}_1 = (1/|S|) \sum_{k=1}^{N} \sum_{\substack{m=1 \\ m \neq k}}^{N} r_{mk} \ell_{mk}^{*} \left(\sum_{j=1}^{c_{mk}} \lambda_{jmk} \right)$$

$$+ (1/|S|) \sum_{k=1}^{N} \sum_{\substack{m=1 \\ m \neq k}}^{N} r_{mk} \sum_{j=1}^{c_{mk}} \lambda_{jmk} (\ell_{(j)}^{mk} - \ell_{mk}^{*})$$

where ℓ_{mk}^{*} is the minimum number of bottleneck channels on any path from m to k. This yields:

$$\hat{\beta}_1 = \beta_L(S, \infty, f) + \text{Dif}(g)$$

where Dif(g) is defined to be the latter term in the equality previous to its appearance. Thus $\text{Dif}(g) = \hat{\beta}_1 - \beta_L(S, \infty, f)$ and is independent of g (providing g solves 4.1.1). This yields the desired condition:

If any $g \in F$ which solves 4.1.1 is such that every active associated chain uses a minimal number of channels which are members of the bottleneck, then the lower bound, $\beta_L(S, \infty, f)$, will be equal to $\hat{\beta}_1$ when the choice of S is correct.

Consult Figs. 4.3.1 and 4.3.2, both of which display very simple cutsets as the bottleneck. It is clear that $\beta_L(S, \infty, f)$ can be made equal to $\hat{\beta}_1$ for both of these networks since any commodity which must use the bottleneck will pass through no more than one bottleneck channel, and all other commodities will avoid the bottleneck entirely.

Figure 4.3.3 displays a situation where it is not clear whether the above condition holds. However, in the solution of the associated linear routing problem generated by the LCH algorithm, the following occurs:

(i) The minimum number of bottleneck channels between nodes 4 and 7 is one, e.g. as evidenced by path (4, 5, 7).

(ii) The chains used for the commodity (4, 7) in the solution of 4.1.1 generated by the LCH algorithm were:

92.6% on path (4, 5, 7)

1.6% on path (4, 6, 7)

5.8% on path (4, 2, 7)

(iii) Both of the latter paths are active, and they use more than the minimal number of bottleneck channels between nodes 4 and 7.

Thus $\beta_L(S, \infty, f)$ cannot be made as great as $\hat{\beta}_1$ when S is chosen as the bottleneck, and the required difference could be evaluated if desired according to the definition of Dif(g).

It follows that how $\beta_L(S, \sigma, f)$ varies as a function of σ, and in particular which value of σ provides the largest lower bound, depends on the problem at hand. Many times, regardless of whether the AL*$(\alpha, \epsilon, \gamma)$ algorithm is "lower bound directed" or "convergence directed", the best value of σ occurs at $\sigma = \infty$ where the bound is exact for a properly chosen S. Often $\beta_L(S, \infty, f)$ (for S chosen as the bottleneck) is not exact but not far below $\hat{\beta}_1$, in which case the "lower bound directed" algorithm might find that a σ near σ_K provides the best bound, while the "convergence directed" algorithm is more likely to find that $\sigma = \infty$ is best. More will be said on this subject in the next section.

4.5 Computational Experience with the AL*$(\alpha, \epsilon, \gamma)$ Algorithm

The computational results of this section are important for more than merely the AL*$(\alpha, \epsilon, \gamma)$ algorithm. This section will confirm the speculations of Chapter II on the relative performance of the EF, TR, and CH algorithms. To this end, the same four examples which were under consideration in Section 4.3 are considered again in this section.

The selection of parameters. The parameters α, ϵ, γ of the AL*$(\alpha, \epsilon, \gamma)$ algorithm are relatively unimportant unless one makes immoderate choices for them. Experience with the algorithm

indicates that the successive problems solved in step 2 of the algorithm should not be drastically different, and yet should differ enough so that the step has some utility. This suggests that γ should be neither particularly close to its lower bound of 1 nor extremely large; using the same reasoning, α should not be near its lower bound of 0 nor its upper bound of 1. Also ϵ should not be chosen unreasonably small, else the minimization of step 2 will be needlessly accurate. A detailed description of the experimentation with these parameters is consequently of little importance. It suffices to say that experimentation led to the chosen values of $\gamma = 5$, $\alpha = 0.6$, and $\epsilon = 0.16$. From this point forward, the parameter list will be dropped from the names of the algorithms, with the implication that the above values are employed.

Relationship to the TR, EF, and CH algorithms. Later in this section, the TR*, EF*, and CH* algorithms will be compared almost as if they were the TR, EF, and CH algorithms. There are several reasons and justifications for this comparison. Since the TR, EF, and CH algorithms all solve the minimum delay problem, there is the difficult question of how heavily the network should be loaded (or how the requirements should be scaled) for a meaningful comparison. Obviously, if the demand on the network is extraordinarily low, the problem can virtually be solved with one shortest route computation. This extreme is not desirable for comparison purposes; however, if the demand is as great as the network will allow, then the algorithms will be compared near their "worst", with nearly as many columns in

the problem as any routing problem will ever require for that network. The latter extreme is precisely that which the TR*, EF*, and CH* algorithms seek. Furthermore, the form of the objective function used in the TR*, EF*, and CH* algorithms is similar in character to that of the expected delay function of Chapter II; i.e. each goes to infinity as the flow in any channel approaches the capacity of the channel. In essence, then, the EF*, TR*, and CH* algorithms solve a sequence of problems very similar to minimum delay problems over a variety of loads on the network.

It is also noted that using the EF* algorithm is an extremely attractive way of starting the EF algorithm (and similarly for the CH and TR algorithms). At least it is the best of the methods that this experimenter has attempted. When σ_K is driven below unity in the EF* algorithm, then the EF algorithm has a feasible (and reasonable) starting point. When the EF algorithm's objective function is complicated, unexpected benefits may occur. A great deal of effort may have been saved by using the objective function of the EF* algorithm until the estimate of the solution is in the correct ballpark.

Discussion of the examples. The examples are presented in turn, along with a discussion of each. With each set of curves, the curve of the fastest linear program from those in Section 5.3 is included. Every algorithm had the same initial flow, although that may not be evident since the curves are not shown until they are in the range of the graph.

Figure 4.5.1 displays the curves for the linear routing problem

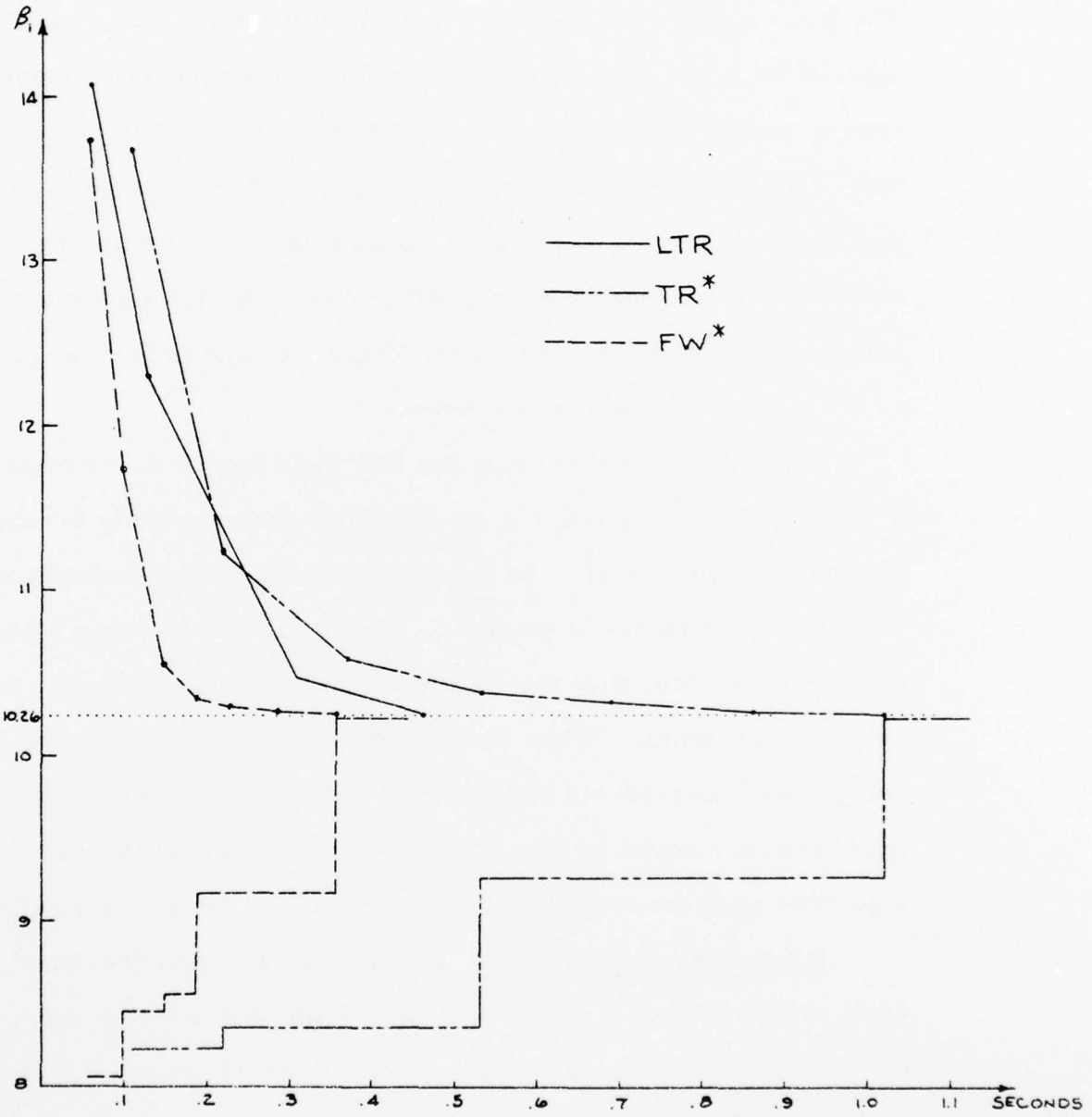


Fig. 4.5.1 The experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.1 for the LTR, TR*, and FW* algorithms. If the curves for the CH* and EF* algorithms had been drawn, they would nearly coincide with that of the TR* algorithm. The curves under the dotted line are the associated lower bounds on $\hat{\beta}_1$ as a function of time.

associated with the ten node network of Fig. 4.3.1. This network has 30 directed arcs or an average of 3 outgoing arcs per node and 33% of the number of arcs of a fully connected network. The FW* algorithm was much faster for this example than the EF*, TR*, and CH* algorithms, although this was not typical of the other examples. The following is a list of the number of columns and the number of storage locations required at the final estimate of the solution (including routing matrices for the EF* algorithm):

	# of columns	# of locations
EF*	17	2210
TR*	45	900
CH*	142	369

It is noted that the TR* and CH* algorithms exceed their "bounds" of Chapter II on the maximum number of columns, although one could pivot out the excess columns if desired.

The lower bound on β_1 became exact at some point in each of the EF*, CH*, TR*, and FW* algorithms for the 10 node problem. This is not to imply, however, that the greatest lower bound could not have been achieved earlier. Much depends on the method of choosing S, and one might try several different methods of choosing S each time the lower bound is calculated. However, a great deal of experimentation with methods for choosing S has not been done. For these examples, only one method was used, and it had the property of being rather conservative (i.e. tending to include too many channels in S)

early in the run, and less so as the algorithm progressed. The intention, which normally was realized, was to obtain a good lower bound at some point in the run of the algorithm without spending a great deal of computation time on it. For these network examples, all of which were solved via "convergence directed" algorithms, the best lower bound on $\hat{\beta}_1$ was invariably found at $\sigma = \infty$.

Figure 4.5.2 displays the curves for the linear routing problem associated with the 26 node network of Fig. 4.3.2. This network has 60 directed arcs or an average of 2.31 outgoing arcs per node and 9% of the maximum number. The remarks of Chapter II on the comparisons of the algorithms indicate that the CH* algorithm would not be as effective as the other algorithms on a very sparsely connected network, and this is reflected in the curves.

The error bound was the least effective on this problem, although the bottleneck is simple (see Appendix E). From Fig. 4.3.2, one can see that arcs (14, 15), (20, 25), and (24, 4) form the bottleneck. It also happens in the solution that the utilization factor of arc (15, 16) turns out to be nearly as great as $\hat{\beta}_1$, and the programs normally chose all four arcs to be the estimate of the bottleneck. With so few arcs actually in the bottleneck, the resultant percentage error in the lower bound from the true value of $\hat{\beta}_1$ is large. A bit of good fortune went into play for the FW algorithm, as it once (incorrectly) chose arcs (15, 16), (20, 25), and (24, 4) as the bottleneck, which nevertheless yielded a fairly good lower bound on $\hat{\beta}_1$. When there are a larger number of arcs in the bottleneck, $\beta_L(S, \sigma, g)$ is not so sensitive to the

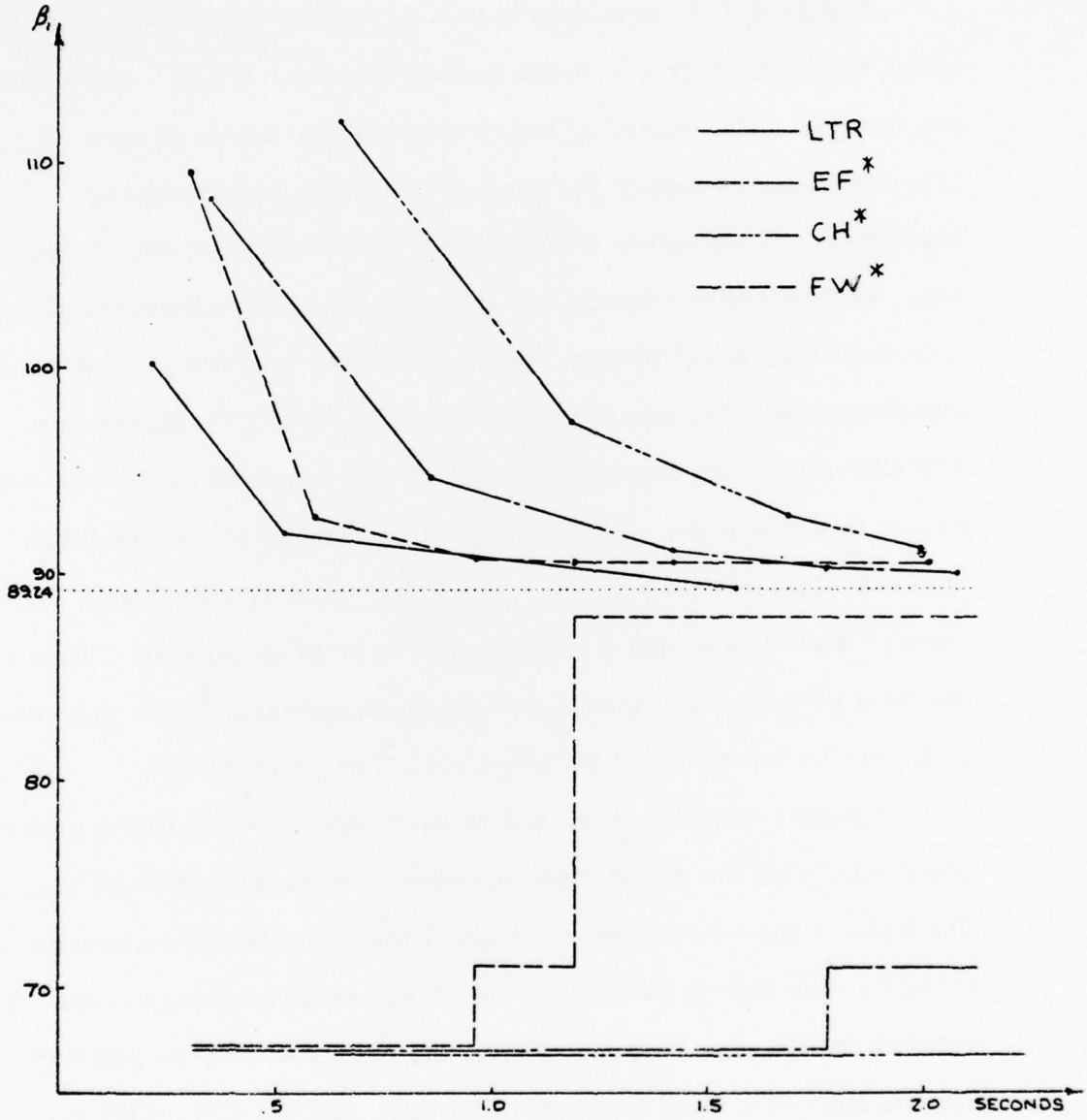


Fig. 4.5.2 The experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.2 for the LTR, EF*, CH*, and FW* algorithms. If the curves for the TR* algorithm had been drawn, they would nearly coincide with that of the EF* algorithm. The curves under the dotted lines are the associated lower bounds on β_1 as a function of time.

choice of S.

Figure 4.5.3 corresponds to the problem associated with the seven node symmetric network depicted in Fig. 4.3.3. Note that for the first time the "knee" of every convergence curve of each AL* algorithm occurs before the knee of the linear programming algorithm. This network is the most fully connected under consideration, with 66% of the number of channels of a fully connected network. It is therefore noted without surprise that the performance of the CH* algorithm timewise was better than the TR* and EF* algorithms. The CH* algorithm required only 61 chains (most of which have only one or two elements), the TR* algorithm required 31 trees (each having 6 elements) and the EF* algorithm used 21 extremal flows (each having 28 elements plus a routing matrix of 49 elements). This was the first network such that $\beta_L(S, \infty, f)$ could not equal $\hat{\beta}_1$ for any choice of S, but the lower bound differed from $\hat{\beta}_1$ by less than $\frac{1}{2}\%$.

Finally, consider Fig. 4.5.4 which corresponds to the problem associated with the eleven node symmetric network shown in Fig. 4.3.4. The knee of the convergence curves of the AL* algorithm is even further to the left of the knee of the linear programming curve. This network is less fully connected than the previous one, having 40% of the channels of a fully connected network, and the EF*, CH*, and TR* algorithms performed nearly equally timewise. At the final estimate of the solution, the CH* algorithm had 161 chains (most of which had no more than 3 or 4 elements), the TR* algorithm had 59 trees (each having 10 elements), and the EF* algorithm had 39 extremal flows

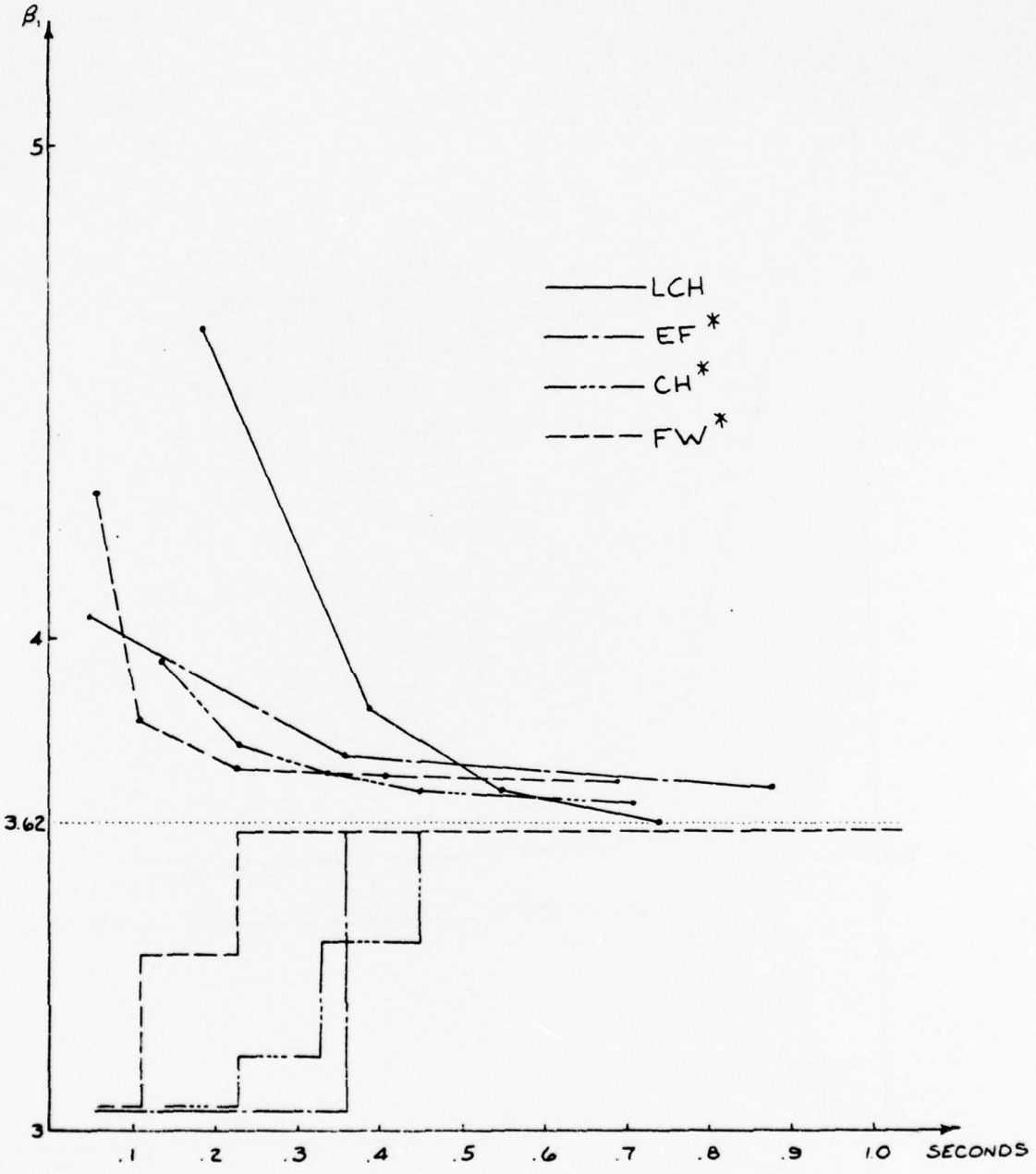


Fig. 4.5.3 The experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.3 for the LCH, EF*, CH*, and FW* algorithms. If the curves for the TR* algorithm had been drawn, they would be in between those of the EF* and CH* algorithms. The curves under the dotted line are the associated lower bounds on $\hat{\beta}_1$ as a function of time.

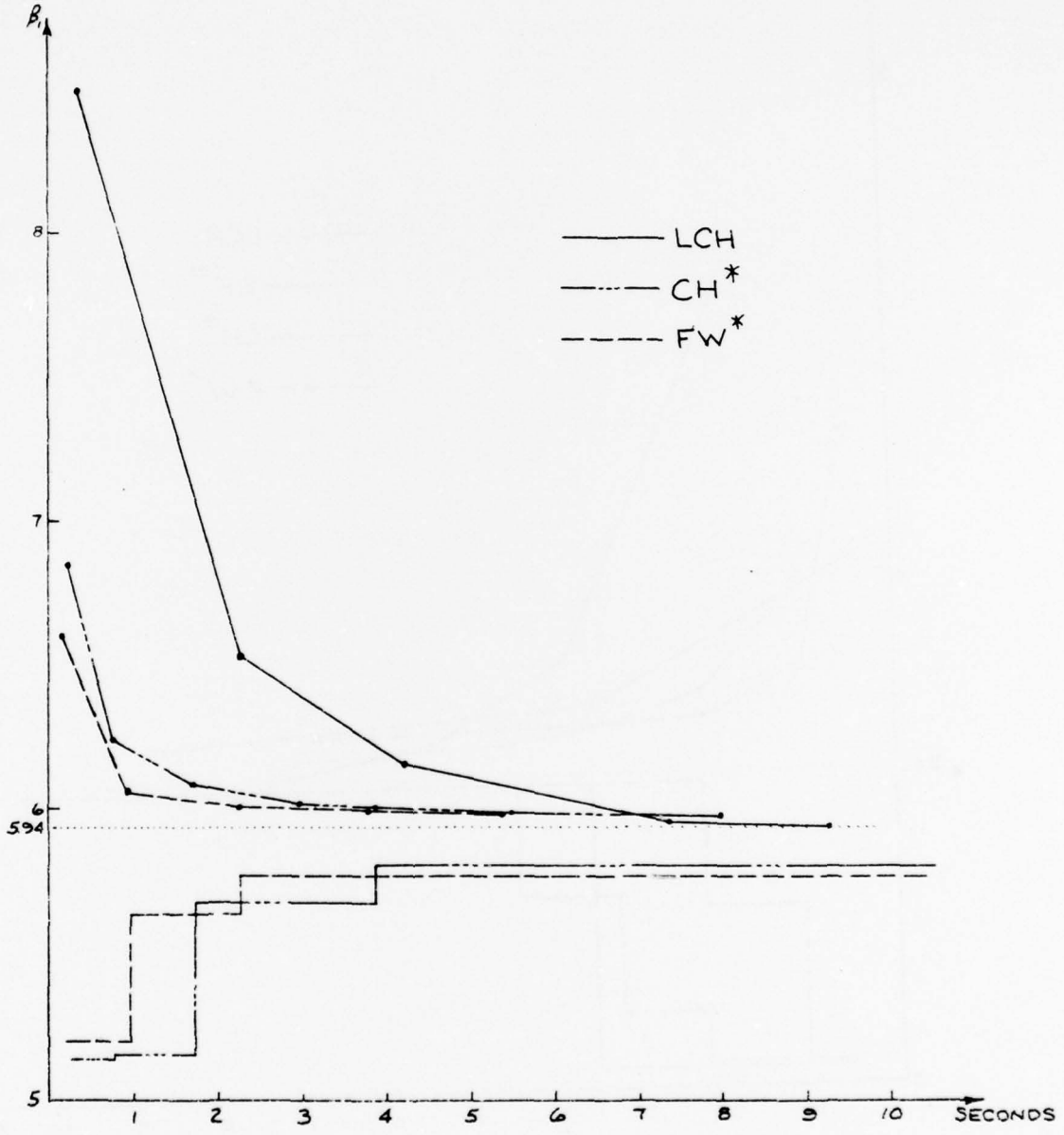


Fig. 4.5.4 The experimental convergence of β_1 of the linear routing problem associated with Fig. 4.3.4 for the LCH*, CH*, and FW algorithms. If the curves for the TR* and EF* algorithms had been drawn, they would nearly coincide with that of the CH* algorithm. The curves under the dotted line are the associated lower bounds on $\hat{\beta}_1$ as a function of time.

(each having 44 elements plus a routing matrix of 121 elements). Once again, the TR* and CH* algorithms exceeded their "bounds" on the bumber of columns, but not by much. Although $\hat{\beta}_1(S, \infty, f)$ could not be made exact, lower bounds about 2% beneath the value of $\hat{\beta}_1$ were achieved.

As the problems became more complicated (in terms of the number of binding constraints in the solution), the trend is very clear. The "knee" of the convergence curves of the AL* algorithms occurs far before the linear program converges to the solution. If one is content with the inaccuracy that exists at the knee of the convergence curve (which is less than about 2% for these problems) and one expects the problem to be complicated, then the nonlinear approach may be the preferred approach. If one wanted to run the programs only until the knee of the convergence is reached, then it is probably in his best interest to try several means of choosing S at the termination in order to get the best lower bound on the objective.

One practical advantage of the nonlinear approach is that it also tends to be searching for a solution which takes into account higher order criteria for the solution. For example, in the LTR solution to the problem associated with Fig. 4.3.2, there are ten active constraints, although only three are required to be active. As one can see from the form of the objective functions for the nonlinear approach, those utilization factors which are not necessarily at $\hat{\beta}_1$ will tend to be driven to a lower value. Thus although the estimate of the solution to the linear routing problem which the nonlinear approach

provides for the first criterion has a higher value for the objective than the linear program, other aspects of the solution as viewed from the higher order criteria may be better (see Appendix D).

CHAPTER V

STATISTICAL LOAD SHARING IN A COMPUTER-COMMUNICATION NETWORK

This chapter deals with the application of the concepts developed in this thesis to the problem of load sharing in a computer-communication network. A set of computers is assumed to be connected via a store and forward network. Each computer receives jobs which it must process and return to the customer. Since the computers are interconnected, a heavily loaded computer has the option of sending jobs elsewhere via the communication network, having the jobs processed at that location, and having the results returned via the network. In this way, the resources of the entire system can be pooled for the overall benefit. The model of this system which is employed in this chapter is that of Wunderlich [WUND75] and McGregor [McGR74].

This chapter is not directed towards providing a large amount of qualitative insight on the load sharing problem. The general subject matter of load sharing is handled in greater depth in Wunderlich's work [WUND76]; however, the details of the algorithm which is applied to the statistical load sharing problem are more complete in this thesis.

5.1 Models and Definitions

The model of a computer used in this study is an M/M/1 queue. This is a simple model of a computer operating in a batch processing mode [KLEI75]. For this model it is assumed that:

1. The input stream of jobs to the jth computer from outside the system is a Poisson process with mean arrival rate λ_j jobs per unit time.

2. The number of operations required per job is distributed as a negative exponential with mean $1/\ell$.

3. The jth computer performs R_j operations per unit time. This means that the service time per job (not including waiting time) is distributed as a negative exponential with mean $1/\ell R_j$.

4. Jobs are processed in a first-come-first-served manner. If the computer is busy when a job arrives, it is queued in an infinite buffer.

The model used for a store and forward communication channel is also an M/M/1 queue. For the communication channel it is assumed that:

1. The length of messages in bits is distributed as a negative exponential with mean $1/\mu_p$ for programs (computer inputs) and mean $1/\mu_r$ for results (computer outputs).

2. The ith communication channel has a capacity C_i bits per unit time so that the time to transmit a message is also distributed as a negative exponential

with mean $1/\mu_p C_i$ or $1/\mu_r C_i$.

3. If a message passes through more than one communication channel, its length is assumed to be chosen independently at each channel through which it passes.

This queueing model of a store and forward communication channel has been extensively studied by Kleinrock [KLEI64].

One final assumption that must be made is that the message lengths of programs, the number of computer operations they require and the message lengths of the results are all independent random variables. This assumption is required to make the problem mathematically manageable.

The performance measure used in this study is the steady state expected time to process a computer job. For a system of N computers, this expected time is given by:

$$T = \sum_{j=1}^{N} \frac{\lambda_j}{\lambda_T} T_j \qquad (5.1.1)$$

$$\text{where } \lambda_T = \sum_{j=1}^{N} \lambda_j$$

The time T_j is the expected time to process a computer job that enters the system at the jth computer. This time includes the computation time required by the job and, if the job is processed by a computer other than the one at which it was submitted, it also includes the communication time required to send the program to the processing

computer and the time to return the results to the point of origin.

5.2 Formulation of the Load Sharing Problem as a Multicommodity Flow Problem

This section considers the formulation and solution of the statistical load sharing problem as a multicommodity flow problem. Presentation of the formulation is best done via a small example. A three-computer network is depicted in Fig. 5.2.1. The network topology for representation as a multicommodity flow problem is shown in Fig. 5.2.2, where each computer is associated with two nodes (an input and an output node), and each physical communication channel is represented by one channel for programs and a second for results. These following definitions are required for the multicommodity formulation:

f_j = average demand on computer j (jobs per unit time).

f_{pi} = average demand on channel i by jobs in program form (jobs per unit time).

f_{ri} = average demand on channel i by jobs in result form (jobs per unit time).

NA = number of communication channels.

Since the only delay associated with the communication channels is assumed to be the queueing delay, the average delay incurred by a job (in program or result form) on channel i may be approximated by:

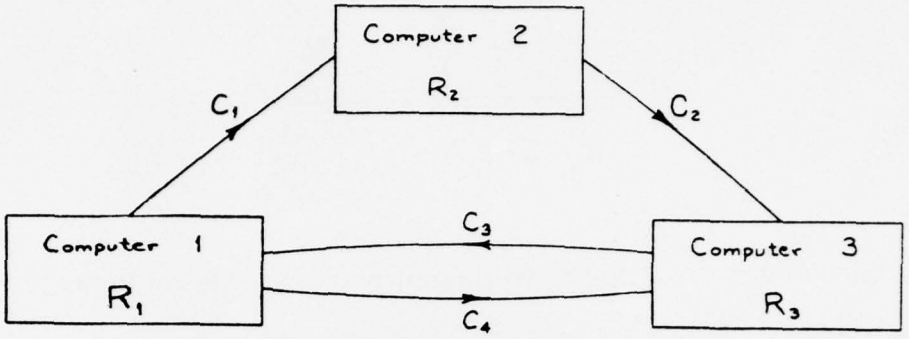


Fig. 5.2.1 A three computer network connected by a communication network.

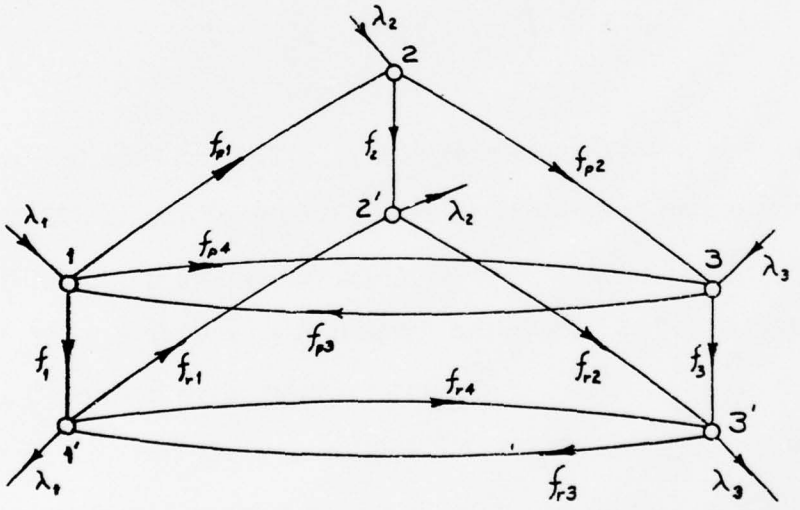


Fig. 5.2.2 The representation of the system of Fig. 5.2.1 for a multicommodity flow problem.

$$\frac{\frac{f_{ri}}{\mu_r} + \frac{f_{pi}}{\mu_p}}{(f_{pi} + f_{ri})\left(C_i - \left(\frac{f_{pi}}{\mu_p} + \frac{f_{ri}}{\mu_r}\right)\right)} \qquad (5.2.1)$$

which agrees with the well-known average delay formula for the M/M/1 queue model of a communication channel in the case $f_{pi} = 0$ or $f_{ri} = 0$ or $\mu_p = \mu_r$.† The average delay of a randomly selected job passing through a system of computers is then approximated by:

$$\tilde{T} = \frac{1}{\sum_{k=1}^{N} \lambda_k} \left\{ \sum_{i=1}^{NA} \frac{\frac{f_{pi}}{\mu_p} + \frac{f_{ri}}{\mu_r}}{C_i - \left(\frac{f_{pi}}{\mu_p} + \frac{f_{ri}}{\mu_r}\right)} + \sum_{j=1}^{N} \frac{f_j}{\ell R_j - f_j} \right\} \qquad (5.2.2)$$

which is a consequence of 5.1.1, 5.2.1, and the fact that the entire system has been modeled as a network of queues [JACK57].

Define f_j^k to be the average demand on computer j due to jobs which enter and leave the system at computer k. That is, $f_j = \sum_{k=1}^{N} f_j^k$. In an analogous manner, also define f_{pi}^k and f_{ri}^k. Then for every $j, k \in \{1, 2, \cdots, N\}$, one may write two conservation of flow equations (which are most easily understood in conjunction with Fig. 5.2.2):

---

† Discussion of this approximation, and of additional conditions under which Equation 5.2.1 is exact, is deferred to Section 5.5.

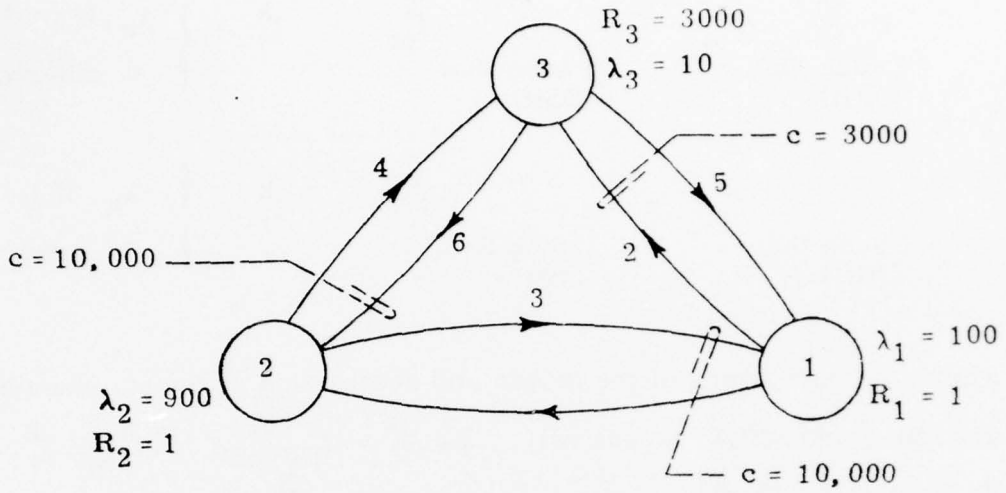
$$\sum_{\substack{i \text{ such that} \\ OR(i) = j}} f_{pi}^k - \sum_{\substack{i \text{ such that} \\ DS(i) = j}} f_{pi}^k + f_j^k = \begin{cases} \lambda_k & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{\substack{i \text{ such that} \\ OR(i) = j}} f_{ri}^k - \sum_{\substack{i \text{ such that} \\ DS(i) = j}} f_{ri}^k - f_j^k = \begin{cases} -\lambda_k & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

where the computers of the origin and destination of the ith channel are OR(i) and DS(i) respectively. Let $f^k = \left[f_{p1}^k, f_{p2}^k, \cdots, f_{pNA}^k, f_{r1}^k, f_{r2}^k, \cdots, f_{rNA}^k, f_1^k, f_2^k, \cdots, f_N^k\right]'$, and define $f = \sum_{k=1}^{N} f^k$. The vector f is said to be an element of the set F if there exists a collection, $\{f^1, f^2, \cdots, f^N\}$, such that (i) $f^k \geq 0$ for every k, (ii) f^k satisfies the above conservation of flow equations for every j and k, and (iii) $\sum_{k=1}^{N} f^k = f$. Then the routing problem may be stated as:

$$\underset{f \in F}{\text{minimize}}\ \tilde{T}(f) \tag{5.2.3}$$

where $\tilde{T}(f) \triangleq \infty$ if $(f_{pi}/\mu_p + f_{ri}/\mu_r) \geq C_i$ for any i or $f_j > \ell R_j$ for any j. If $f^{\#}$ solves 5.2.3 and $f_{pi}^{\#} = 0$ or $f_{ri}^{\#} = 0$ for every $i \in \{1, 2, \cdots, NA\}$, then the expression for the average delay becomes exact at $f^{\#}$ subject to the conditions of the model. For many practical problems this will be the case; however, the many problems involving links which are not duplex and some problems with all duplex links (see Fig. 5.2.3), the expression for delay remains an approximation.



Solution Generated by Multicommodity Flow Program				
Computer of job entrance	Pr {job uses this path}	Flow Path		
		Program	Computer	Result
1	1.0	1, 4	3	5
2	0.898	4	3	6
2	0.102	4	3	5.1
3	1.0	-	3	-

Fig. 5.2.3 Example of a network in which both programs and results use the same communication channel (channel 1) in the optimum solution to the minimum delay problem.

5.3 An Efficient Algorithm to Solve the General Computer Load Sharing Problem

At this point, an objective function has been derived which is convex and has a gradient with all components non-negative over F so that a variety of algorithms may be directly applied to solve 5.2.3. Decomposition techniques, such as the extremal flow (EF) algorithm of Cantor and Gerla [CANT74] are particularly attractive. This section presents modifications to the EF algorithm which produce an efficient algorithm for the load sharing problem. Initially, it is noted that $\partial\tilde{T}(f)/\partial f_{ri} = (\mu_p/\mu_r)(\partial\tilde{T}(f)/\partial f_{pi})$, and this fact is exploited in both the master problem and the subproblem of the EF algorithm to reduce the number of arcs and the number of nodes which are considered explicitly by the algorithm.

The master problem of the EF algorithm is:

$$\text{minimize } \tilde{T}\left(\sum_{i=1}^{M} \alpha_i \varphi^{(i)}\right)$$

$$\text{subject to } \sum_{i=1}^{M} \alpha_i = 1, \text{ and } \alpha_i \geq 0$$

where $\{\varphi^{(1)}, \varphi^{(2)}, \cdots, \varphi^{(M)}\}$ is a generated set of extremal flows of the set F. The fundamental operation necessary to carry out the minimization is the calculation of $\frac{\partial}{\partial \alpha_j} \tilde{T}\left(\sum_{i=1}^{M} \alpha_i \varphi^{(i)}\right)$. If $\nabla\tilde{T}(f) = [\partial\tilde{T}(f)/\partial f_{p1}, \partial\tilde{T}(f)/\partial f_{p2}, \cdots, \partial\tilde{T}(f)/\partial f_{pNA}, \partial\tilde{T}f)/\partial f_{r1}, \partial\tilde{T}(f)/\partial f_{r2}, \cdots, \partial\tilde{T}(f)/\partial f_{rNA}, \partial\tilde{T}(f)/\partial f_1, \partial\tilde{T}(f)/\partial f_2, \cdots, \partial\tilde{T}(f)/\partial f_N]'$, $\langle , \rangle$ denotes the usual inner product in Euclidian space, and $f^* = \sum_{i=1}^{N} \alpha_i \varphi^{(i)}$ (where f^* is the current estimate of $f^{\#}$), then

$$\frac{\partial \tilde{T}(f^*)}{\partial \alpha_j} = \langle \nabla \tilde{T}(f^*), \varphi^{(j)} \rangle$$

$$= \sum_{i=1}^{NA} \frac{\partial \tilde{T}(f^*)}{\partial f_{pi}} \varphi_{pi}^{(j)} + \sum_{i=1}^{NA} \frac{\partial \tilde{T}(f^*)}{\partial f_{ri}} \varphi_{ri}^{(j)} + \sum_{i=1}^{N} \frac{\partial \tilde{T}(f^*)}{\partial f_i} \varphi_i^{(j)}$$

$$= \sum_{i=1}^{NA} \frac{\partial \tilde{T}(f^*)}{\partial f_{pi}} \left(\varphi_{pi}^{(j)} + \frac{\mu_p}{\mu_r} \varphi_i^{(j)} \right) + \sum_{i=1}^{N} \frac{\partial \tilde{T}(f^*)}{\partial f_i} \varphi_i^{(j)}$$

$$= \langle \nabla \tilde{T}(\tilde{f}^*), \tilde{\varphi}^{(j)} \rangle .$$

The last equality follows from the definition $\tilde{f} = [\tilde{f}_1, \tilde{f}_2, \cdots \tilde{f}_{NA}, f_1, f_2, \cdots, f_N]'$, where $\tilde{f}_i = f_{pi} + (\mu_p/\mu_r) f_{ri}$ and the extremal flows, $\tilde{\varphi}^{(j)}$, are similarly defined. As a consequence, all storages and computations may be made in terms of vectors of the form $\tilde{f}$ rather than of f, with consequent reductions in computation time and memory.

In the subproblem which generates a new extremal flow, one solves the linear program

$$\underset{f \in F}{\text{minimize}} \langle \nabla \tilde{T}(f^*), f \rangle$$

and the technique for solution is via a shortest route algorithm. The distance associated with the ith channel is $\partial \tilde{T}(f^*)/\partial f_{pi}$ and $\partial \tilde{T}(f^*)/\partial f_{ri}$, respectively, for the unprimed and primed subnetworks as shown in Fig. 5.2.2. The unprimed nodes are those at which jobs enter the system, and the primed nodes are those at which results depart. One may note that there is no path between unprimed nodes which passes through primed nodes and vice versa. Thus the shortest route

computation of the subproblem may be done first over the network of primed nodes, then over the unprimed nodes, and finally combined. Further, the network distances between the primed nodes only differ by a multiplicative constant from the network distances between the corresponding unprimed nodes, so that the shortest routes may be found over the unprimed network with the shortest routes over the primed network being implicit. Ultimately, only the shortest routes between j and j' for every $j \in \{1, 2, \cdots, N\}$ are required (since they are the only origin-destination pairs with a nonzero demand for a flow of jobs), and each may be found via the solution of

$$\underset{1 \le k \le N}{\text{minimize}} \; (d^*_{jk} + d_{kk'} + d^*_{k'j'}) \qquad (5.3.1)$$

where d^*_{jk} is the distance of the shortest j to k path, $d_{kk'} \triangleq \frac{\partial \tilde{T}(f^*)}{\partial f_k}$, and $d^*_{k'j'} = (\mu_p/\mu_r) d^*_{kj}$. Thus the number of nodes explicitly represented is reduced by a factor of two for the subproblem. As shortest route computation time may grow as fast as the number of nodes cubed, this represents up to a factor of eight reduction in shortest route computation time (less the order N^2 operations of 5.3.1, which generally should be small in comparison to the reduction).

The final modification to the EF algorithm is to base the decomposition on chains rather than extremal flows (each extremal flow is a superposition of N chains). This provides greater freedom in finding downhill directions resulting in fewer shortest route

computations, and memory requirements are reduced as chains can be stored compactly. The details of this alteration were indicated in Chapter II. This load sharing algorithm based on a chain decomposition is denoted the LSCH algorithm.

In conclusion of this section, the following remarks contrast the LSCH algorithm with that of McGregor and Boorstyn [McGR75] who devised the only other known algorithm for the problem:

(i) The LSCH algorithm is free of the constraint that results must travel the reverse route of the respective program. This leads to lower values of the objective and allows a more general communication network.

(ii) An upper bound to the absolute error of any estimate of the solution to 5.2.3 is a by-product of the LSCH algorithm, which allows control over the necessary running time of the algorithm for any desired accuracy.

(iii) The routing strategy is not derived from the solution, but is explicitly the solution itself, i.e., the weight of each chain is the fraction of jobs arriving at the source of the chain which should take the route described by the chain through the network.

5.4 An Example of Load Sharing

A small example of the use of this algorithm is presented. The example is of a ten-computer system for each of which $R_i = 10^6$ steps/sec. and $\ell = 4 \times 10^7$ steps/job. The communication network is shown in Fig. 5.4.1 and consists of 30 directed channels, each having capacity 9.6 Kb/sec. The mean lengths of messages are $1/\mu_p = 50$ Kb and $1/\mu_r = 500$ Kb. A randomly generated set of input rates to the computers with and without load sharing are given in Table 5.4.1. These rates, together with the flows of programs and results shown in Fig. 5.4.1, define the statistical load sharing policy. Note that due to the large ratio of μ_p/μ_r, the flows of programs and results are not symmetric. In this case, in fact, no communication link carries both programs and results. The solution was obtained using the algorithm of Section 5.3 and is guaranteed to be within 1% of the optimum load sharing policy. This solution required 0.3 sec of computation time on an IBM 370-168 using a FORTRAN G1 complied program. The shortest route computation times were kept minimal by the use of the decomposition algorithm of Chapter III.

5.5 Additional Remarks on the Approximations of Section 5.2

This section serves two principal purposes. The first is to expand on the rationale underlying the approximations of Equations 5.2.1 and 5.2.2, and the second is to indicate the magnitude of the errors due to these approximations. The latter is accomplished through several mechanisms. Namely, the error in 5.2.1 is derived,

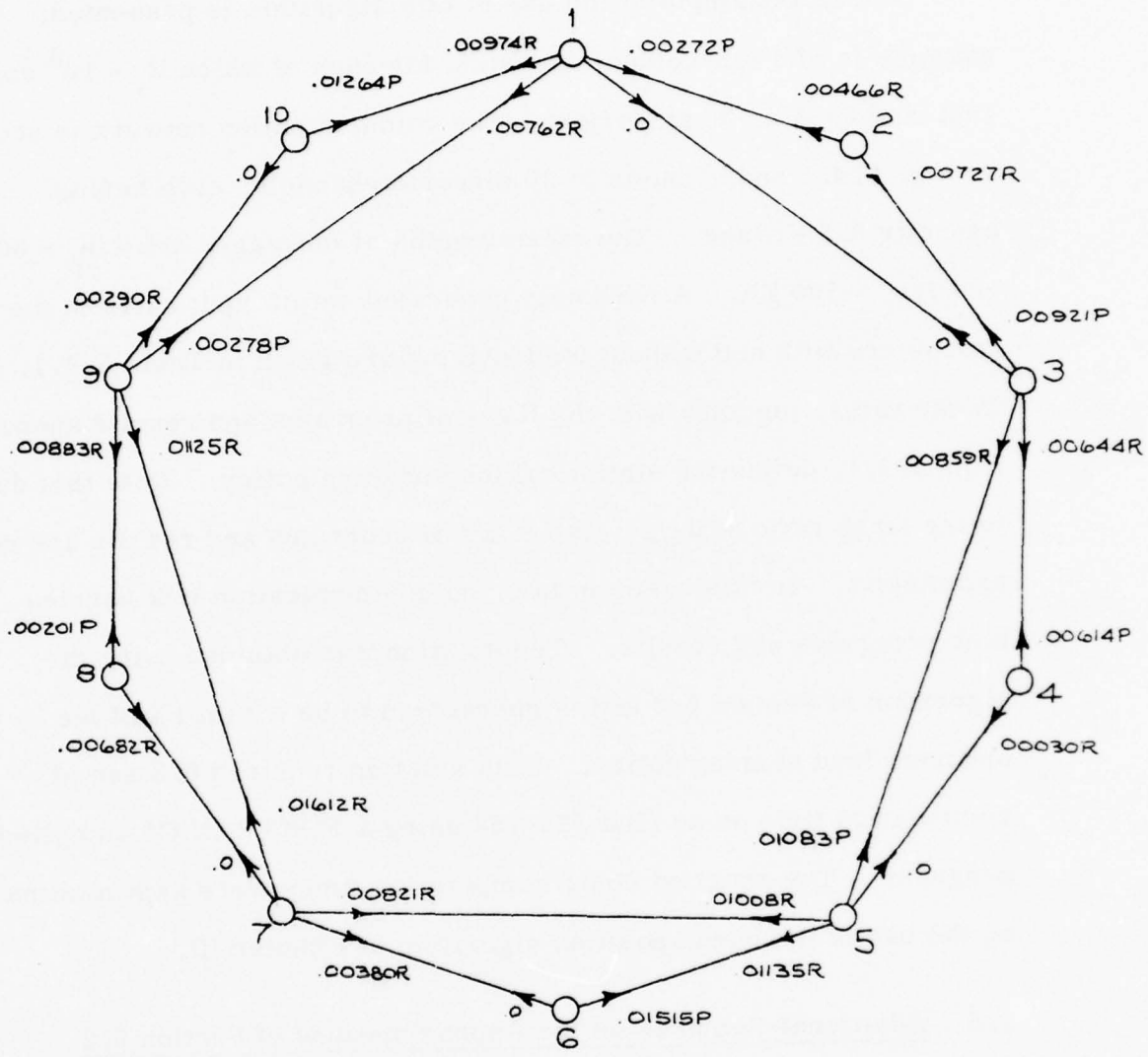


Fig. 5.4.1 Example of ten computer system using load sharing. Values of average job rates on each communication channel in the near optimal solution of the minimum average delay problem are depicted (e.g. 0.00821 jobs/sec in program form on channel (7, 5) and 0.01008 jobs/sec in result form on channel (5, 7)).

the nature of the error in 5.2.2 is discussed, and the changes in the assumptions which make 5.2.1 and 5.2.2 exact are presented.

In Section 5.2, Equation 5.2.1 was proposed as an approximation to the average delay (waiting time plus service time) experienced by a job in passing through the ith channel. In essence, one postulated two independent Poisson input streams to the channels: (i) the program stream with arrival rate f_{pi} and service rate $\mu_p C_i$, and (ii) the result stream with arrival rate f_{ri} and service rate $\mu_p C_i$. Equivalently, one may model these two streams as one Poisson stream with arrival rate $(f_{pi} + f_{ri})$ and a probability density function, $p_t(t_o)$, of the service time, t, for a randomly selected arrival given by:

$$p_t(t_o) = \begin{cases} \frac{f_{pi}}{f_{pi}+f_{ri}} \mu_p C_i \exp(-\mu_p C_i t_o) + \frac{f_{pi}}{f_{pi}+f_{ri}} \mu_r C_i \exp(-\mu_r C_i t_o) & \text{for } t_o \geq 0 \\ 0 \quad \text{otherwise} \end{cases}$$

One can see that the assumptions imply that the channel behaves as an M/G/1, first-come-first-serve (FCFS) queueing system.

The average delay through this system is derived as [KLEI76]:

$$\frac{(f_{pi}+f_{ri})E(t^2)}{2(1-(f_{pi}+f_{ri})E(t))} + E(t)$$

or equivalently

$$\frac{\frac{f_{pi}}{\mu_p} + \frac{f_{ri}}{\mu_r} + \frac{f_{pi}f_{ri}}{C_i}\left(\frac{1}{\mu_p} - \frac{1}{\mu_r}\right)^2}{(f_{pi}+f_{ri})\left(C_i - \left(\frac{f_{pi}}{\mu_p} + \frac{f_{ri}}{\mu_r}\right)\right)} \qquad (5.5.1)$$

Upon a comparison of 5.5.1 to 5.2.1, it is clear that 5.2.1 is a good approximation to 5.5.1 when the third term in the numerator of 5.5.1 is small compared to the sum of the first two terms. Although that is true in some cases, as when $\mu_p \approx \mu_r$, there are cases in which the third term is much larger than the sum of the first two terms. For example, take $C_i = 1$, $\mu_r = 0.1$, $f_{ri} = 0.04$, $\mu_p = 10$, and $f_{pi} = 4$; in this case, the third term is $(0.16)(9.9)^2 = 15.68$ and the sum of the first two terms is $0.4 + 0.4 = 0.8$.

Now concentrating on Equation 5.2.2, it can be understood to be an approximation for two reasons. The first contributor to the inexactness of 5.2.2 is the approximation of 5.5.1 by 5.2.1, and the second is the imprecise application of the theory of networks of queues. From an equivalent point of view, the ith channel is assumed to behave as an M/M/1 queueing system with service rate $C_i(f_{pi} + f_{ri})/(f_{pi}/\mu_p + f_{ri}/\mu_r)$. This single assumption leads to the approximation designated 5.2.1, and it also permits the application of the theory of networks of M/M/1 queues in order for 5.2.2. to result.

Strictly speaking, the form of 5.2.2 is founded on the theory of networks of queues which allows treatment of each queue in the system as if it were independent of all other queues in the system, but only in the case that each queueing center has a special characteristic. In

particular, if the input stream to any given queueing center is a Poisson process, then the departure process must also be Poisson [KOBA77, MUNT73]. However, a channel behaving as an M/G/1 FCFS queueing system is known not to have this property [KLEI76], causing difficulties in the justification of 5.2.2. Therefore, not only are certain terms in 5.2.2 inexact, but the entire form of 5.2.2 is an approximation, so that there would seem to be no simple manner in which to evaluate the associated error. Of course, the problems of approximations vanish when the solution has a certain property, i.e. an absence of channels which are used by both programs and results.

On the other hand, one could have altered the manner in which the channel operates, and thereby avoided all of these issues. In such a case, 5.2.1 and 5.2.2 would be exact results. The required change is to operate the channels in a preemptive-resume last-come-first-serve (LCFS) mode or in a round robin (RR) capacity sharing mode [BASK75]. In actual implementation, both techniques require more complex nodal processing. On the other hand, there are some beneficial effects. By giving short messages a higher priority, one has attained generally lower expected delays (evidenced by the comparison of 5.5.1 with 5.2.1). Also, the analysis and predictions of delays are exact subject only to the conditions and assumptions of Section 5.1.

Computer	λ_i (jobs/sec)	Processing rate of jobs at each computer with load sharing (jobs/sec)
1	.00400	.01671
2	.00339	.01532
3	.00942	.01718
4	.02428	.01814
5	.00594	.01847
6	.03490	.01975
7	.04383	.01949
8	.02103	.01902
9	.00267	.01803
10	.03111	.01847

Table 5.4.1 Computer input rates for example shown in Fig. 9. The expected job time T for this example is infinite before load sharing and 230.2 sec/job after load sharing.

CHAPTER VI

CONCLUSIONS

This thesis has been concerned with the general routing problem in computer-communication networks. One of the best known algorithms which solves the static minimum average delay problem is the EF algorithm. That algorithm has been modified to make the routing information implicit in the notation, rather than requiring the maintenance of an explicit routing matrix for each column involved in the solution. The two modifications which were presented, namely the TR and CH algorithms, were shown to be more compact in terms of memory, while retaining the rapid convergence properties of the EF algorithm. Either algorithm would seem to be preferable to the EF algorithm for most networks; generally the TR algorithm would seem to be preferable to the CH algorithm for sparse networks, while the opposite is true for densely connected networks.

Chapter III describes two new shortest route algorithms, the NXN and IHU algorithms, which are well suited to the subproblems of the various routing algorithms of this thesis. It is characteristic of those routing algorithms that (i) the shortest route calculations are done many times for the same network topology, and (ii) the storage of the entire distance matrix is not burdensome compared to the other memory requirements. Thus a modest investment in the decomposition of the network, and the allocation of a small amount of memory to record the information, can result in substantial reductions in shortest

route computation time. The NXN algorithm is most effective for networks which are planar and sparsely connected, and the IHU algorithm is most effective for those networks of a more particular structure described in Chapter III.

In Chapter IV, a linear routing problem was defined which minimizes the maximal utilization factor of any channel in the network. Such a routing policy would seem desirable in a computer-communication network, especially when considering the finite buffer size at each node of the network. This follows from the observation that minimizing the maximum channel utilization corresponds, approximately, to minimizing the maximum probability that a packet is lost at any given node due to lack of buffer facilities. The minimax routing problem was solved initially as a linear program via decomposition techniques analogous to those of the nonlinear algorithms of Chapter II. It was generally found that the LCH algorithm was the most effective on those problems which are difficult to solve, and hence on those problems where efficiency is the most important.

These linear algorithms were then compared to a nonlinear algorithmic approach based on the algorithms of Chapter II. If one is content with the inaccuracy of the nonlinear algorithms (generally less than 2% of the objective) at the "knee" of the convergence curves, then they are preferable to the linear algorithms as the complexity of the problem increases. In particular, the nonlinear algorithms automatically take partial account of higher order criteria for the solution; and they more easily solve a perturbation of a problem, using the

previous estimate of the solution as a starting point, than the linear programs do.

Finally, in Chapter V, an example of the application of the ideas of the thesis to computer load sharing was developed. Although the problem at first glance may not seem to be a multicommodity flow problem in the sense of Chapter I, it can be modeled as such. The example serves to indicate how, for a particular problem, the special structure of that problem may be utilized to increase the efficiency of the algorithms presented in this thesis.

Suggestions for future research. This final chapter will conclude with selected suggestions for future research. First, in Chapter II, the range of algorithms under consideration was rather narrowly constrained. For example the question of whether to centralize or distribute the algorithms for routing was not asked, and remains unanswered. To define the "robustness" of an algorithm in such a way as to permit a meaningful comparison of distributed and centralized algorithms for a fixed allocation of the resources of the network, would have a great impact on the computer-communication network area.

Second, there may be ways to improve the convergence of the algorithms of Chapter II when they are in the neighborhood of the solution. Finding downhill directions for subgroups of commodities, rather than for the entire network, may be one direction of research. Insofar as how commodities should be grouped, the only limitations seem to be those imposed by one's perception of the cause of the

decrease in experimental convergence rate of the various algorithms.

Most extensions of work in Chapter II would have direct application of Chapter IV. But Chapter IV has some problems which are unique. In particular, improved procedures for estimating the bottleneck, in order to improve the lower bound of Section 4.4, would be of interest. There may exist characteristics of the bottleneck channels which could be exploited in order to narrow the choice of channels to be included in the estimate.

Appendix B mentions "stability" of a network without attempting a definition of the term. To define stability and to pursue its implications could have great impact on the problem of dynamic network control. Depending on the direction which such a study might take, some of the algorithms of this thesis might have great utility in finding the routing policies which maximize the stability of a network.

APPENDIX A

SELECTED PROOFS

This appendix contains the proofs of selected remarks within the body of the report where continuity of thought prevented a more rigorous discussion.

Theorem 2.1.A: The absolute delay error, $T(f^*) - T(f^\#)$, at the completion of a restricted master problem is upper bounded by $-\text{Rel Cost}(\varphi_{(j+1)})$, where f^* is the current estimate of $f^\#$, $f^\#$ is the solution to 2.1.1, and $\varphi_{(j+1)}$ is the extremal flow generated by the associated subproblem.

Proof: Since T(f) is a convex function, then it lies above the best linear approximation of T(f), i.e.

$$T(f) \geq T(f^*) + \langle \nabla T(f^*), f-f^* \rangle$$

If $f \in F$, then

$$T(f^*) - T(f) \leq - \langle \nabla T(f^*), f-f^* \rangle \leq - \text{Rel Cost}(\varphi_{(j+1)})$$

by the definition of $\text{Rel Cost}(\varphi_{(j+1)})$. In particular, let $f = f^\#$, which proves the theorem.

Theorem 2.2.A: Any flow f which can be expressed as a convex combination of trees, can be expressed as a convex combination of at most NA + N of them.

Proof: If f can be expressed as a convex combination of trees, then

there exists $\{\lambda_{ik}\}$ such that $f = \sum_{k=1}^{N} \sum_{i=1}^{t_k} \lambda_{ik} \varphi_{(i)}^{k}$, $\sum_{i=1}^{t_k} \lambda_{ik} = 1$, and $\lambda_{ik} \geq 0$, where the notation is in agreement with that of Section 2.2. Using linear programming terminology, $\{\lambda_{ik}\}$ is a feasible solution to a system of NA + N linear equations which are shown in Equation 4.1.3; therefore, the system has a basic feasible solution, $\{\lambda'_{ik}\}$, where there are at most NA + N nonzero elements [DANT63]. (Thus, if the number of trees in the basis of the TR algorithm exceeds NA + N, then the basis may be reduced to NA + N elements by pivot operations which are fundamental in linear programming.)

Theorem 2.3.A: The direction, Δf, as defined in section 2.3, is a feasible downhill direction.

Proof: Recall the form of f^*, i.e. $f^* = \sum_{k=1}^{N} \sum_{i=1}^{j_k} \lambda_{ik} \varphi_{(i)}^{k}$ such that $\sum_{i=1}^{j_k} \lambda_{ik} = 1$, $\lambda_{ik} \geq 0$. Any flow, f, which can be expressed in the same form (as a convex combination of the trees in the basis), is said to be feasible with respect to the current basis. Δf is a feasible direction if there exists $\delta > 0$ such that $f^* + \delta \Delta f$ is feasible with respect to the current basis. Assume, momentarily, that every $\lambda_{ik} > 0$; then by construction, $f^* + \delta \Delta f = \sum_{k=1}^{N} \sum_{i=1}^{t_k} (\lambda_{ik} + \delta \Delta \lambda_{ik}) \varphi_{(i)}^{k}$. From the definition $\Delta \lambda_{ik} = -\partial T(f^*)/\partial \lambda_{ik} + (1/t_k) \sum_{m=1}^{N} \partial T(f^*)/\partial \lambda_{mk}$, the condition $\sum_{i=1}^{t_k} (\lambda_{ik} + \delta \Delta \lambda_{ik}) = 1$ is maintained for any δ. Also, if $0 \leq \delta \leq \min_{\Delta \lambda_{ik} < 0} (-\lambda_{ik}/\Delta \lambda_{ik})$, then $(\lambda_{ik} + \delta \Delta \lambda_{ik}) \geq 0$. Thus, Δf is a feasible direction. Notice that the restriction $\lambda_{ik} > 0$ may be relaxed to $\lambda_{ik} \geq 0$ if $\Delta \lambda_{ik} > 0$. Thus a newly added $\varphi_{(i)}^{k}$ from a subproblem poses no difficulty even though

$\lambda_{ik} = 0$, since $\Delta\lambda_{ik} > 0$.

Δf is a downhill direction if $\frac{\partial}{\partial\beta} T(f^* + \beta\Delta f)\Big|_{\beta=0} < 0$. Furthermore,

$$\frac{\partial T}{\partial\beta}(f^* + \beta\Delta f)\Big|_{\beta=0} = \langle \nabla T(f^*), \Delta f\rangle = \sum_{k=1}^{N} \sum_{i=1}^{t_k} \Delta\lambda_{ik} \langle \nabla T(f^*), \varphi_{(i)}^k\rangle$$

$$= \sum_{k=1}^{N} \sum_{i=1}^{t_k} \left(-\frac{\partial T(f^*)}{\partial \lambda_{ik}} + \frac{1}{t_k} \sum_{m=1}^{N} \frac{\partial T(f^*)}{\partial \lambda_{mk}} \right) \frac{\partial T(f^*)}{\partial \lambda_{ik}}$$

$$= \sum_{k=1}^{N} \left\{ \sum_{i=1}^{t_k} - \left(\frac{\partial T(f^*)}{\partial \lambda_{ik}}\right)^2 + \frac{1}{t_k} \left(\sum_{i=1}^{t_k} \frac{\partial T(f^*)}{\partial \lambda_{ik}} \right)^2 \right\} \leq 0$$

where the last inequality follows from the Schwartz inequality. Equality occurs only if $\partial T(f^*)/\partial \lambda_{1,k} = \partial T(f^*)/\partial \lambda_{2,k} = \cdots = \partial T(f^*)/\partial \lambda_{t_k k}$ for every $k \epsilon \{1, 2, \cdots, N\}$, which, in turn, implies f^* is the solution to the current master problem of the TR algorithm in which case there can be no feasible downhill direction. Q. E. D.

Theorem 2.4.A: Any flow f which can be expressed as a convex combination of chains, can be expressed as a convex combination of at most NA+N(N-1) of them.

Proof: If f can be expressed as a convex combination of chains, then there exists $\{\lambda_{imk}\}$ such that $f = \sum_{m=1}^{N} \sum_{\substack{k=1 \\ k \neq m}}^{N} \sum_{i=1}^{c_{mk}} \lambda_{imk} \varphi_{(i)}^{mk}$, $\sum_{i=1}^{c_{mk}} \lambda_{imk} = 1$, and $\lambda_{imk} \geq 0$ where the notation is in agreement with that of Section 2.4. The proof follows the same lines as Theorem 2.2.A after one notes that the set, $\{\lambda_{imk}\}$, is a feasible solution to a system of NA+(N-1)N linear equations analogous to those shown in Equation 4.1.3.

Theorem 4.4.A: The lower bound, $\beta_L(S,\sigma, f)$ as defined in Section 4.4, can converge to $\hat{\beta}_1$.

Proof: Whether or not $\beta_L(S,\sigma, f)$ converges to β_1 depends on the methodology employed in the selection S and σ of $\beta_L(S,\sigma, f)$ and ϵ of AL*$(\alpha, \epsilon, \gamma)$. In this proof, it is shown that there exists at least one set of choices which implies the convergence of $\beta_L(S,\sigma, f)$ to $\hat{\beta}_1$.

Assume that the lower bound is computed upon the completion of step 2 of the AL*$(\alpha, \epsilon, \gamma)$ algorithm on each iteration, and that

(i) $\sigma = \sigma_K$

(ii) $f = f^K$

(iii) S is chosen as the set of all the channels to the network.

(iv) The parameter ϵ is chosen such that $0 < \epsilon < 1$.

Under these circumstances, the lower bound is denoted β_L^K, and

$$\beta_L^K \geq \sigma_K - \frac{NA\,\sigma_K}{(1-\epsilon)\sum_{i=1}^{NA} \frac{1}{1-\rho_i^K/\sigma_K}}$$

$$\geq \sigma_K - \frac{NA\,\sigma_K}{(1-\epsilon)\frac{1}{1-\beta_{1'}(f^K)/\sigma_K}}$$

$$\geq \sigma_K - \frac{NA}{(1-\epsilon)}\,(\sigma_K - \beta_1(f^K))$$

Now it is known from the proof of the convergence of the AL*$(\alpha, \epsilon, \gamma)$ algorithm (Equation 4.4.1) that:

$$\sigma_K - \hat{\beta}_1 \leq (1 - r)^{K-1} (\gamma\beta_1(f^0) - \hat{\beta}_1)$$

where $r = \alpha/(NA(1+\epsilon))$. Since $\sigma_K - \beta_1(f^K) \leq \sigma_K - \hat{\beta}_1$, then

$$\beta_L^K \geq \hat{\beta}_1 - \frac{NA}{(1-\epsilon)}(1 - r)^{K-1} (\gamma\beta_1(f^0) - \hat{\beta}_1)$$

The last inequality and the fact $\beta_L^K \leq \hat{\beta}_1$ imply that β_L^K converges at least geometrically to $\hat{\beta}_1$. Q. E. D.

APPENDIX B

PERTURBATIONS AND THE ROUTING ALGORITHMS

This appendix demonstrates the capability of routing algorithms to resolve the minimum average delay problem for a perturbation of a demand matrix for which the problem has already been solved. As noted in Chapter II, such capability is inherent in the EF, TR, and CH algorithms although the FW algorithm does not have this ability. There is no reason to believe that the EF, TR and CH algorithms should behave very differently from one another on a perturbation of a previously solved problem; hence, for convenience, the methodology for handling perturbations (as well as the example of its use) is presented only for the TR algorithm.

As a by-product of the investigation of perturbations of the requirement matrix, one can obtain a qualitative feel for the proper time intervals between updates of the routing policy for a quasistatic implementation. This, however, would need to be conditioned on the probabilistic nature of the elements of the requirement matrix, and only one model is considered here. The reasons that the particular model was chosen may be more clear after reading Appendix C.

A model of the requirement matrix. As a preliminary, the model of the perturbations of the demand matrix is presented. The requirement matrix will be defined at discrete times, $t \in \{0, 1, 2, \cdots\}$, and will be denoted $R(t) = [r_{ij}(t)]$. The following specify the demands, $r_{ij}(t)$:

(i) $r_{ij}(t_1)$ and $r_{mk}(t_2)$ are independent random variables providing $i \neq m$ or $j \neq k$ for every t_1 and t_2.

(ii) If $i = j$, $r_{ij}(t) = 0$. Otherwise, $r_{ij}(t)$ has the density function of the absolute value of a zero mean Gaussian random variable with a variance of 3.

(iii) If $i \neq j$, then $r_{ij}(t)$ given $r_{ij}(t-1)$ has the density function of $|r_{ij}(t-1) + \alpha_t x_{ij}(t)|/(1+\alpha_t^2)^{\frac{1}{2}}$, where $x_{ij}(t)$ is a zero mean Gaussian random variable with a variance of 3 which is independent of all other random variables in the network (including $x_{ij}(\tau)$ if $\tau \neq t$, and where $\alpha_t > 0$. It is not hard to verify that (ii) and (iii) are not in conflict.

The problem. The problem which is to be solved is described below. The network is the seven node symmetric network of Fig. 4.3.3. Every channel is assumed to have capacity 5.5. The requirement matrix, $R(t) = [r_{ij}(t)]$, at $t = 0$ is the one which would be specified by the rules of Appendix C for a static problem. The minimum delay problem

$$\min_{f \in F} \frac{1}{\gamma} \sum_{i=1}^{NA} \frac{f_i}{5.5 - f_i}$$

is solved such that the upper bound on the error is less than 2%. Then R(1) is generated according to specification (iii) above. The current estimate of the solution, i.e. $f^*(0) = \sum_{k=1}^{N} \sum_{j=1}^{t_k} \lambda_{jk} \varphi^k_{(j)}$, is modified in

the following way: the λ_{jk} remain fixed, and the $\varphi^{k}_{(j)}$ are changed as if R(1) were the requirement matrix when they were generated. This is similar (but not equivalent) to keeping the routing policy fixed while the requirement matrix has changed.

Clearly, the new resulting flow, f(1), is a member of the new set F; however, the flow may not be feasible, in which case appropriate measures are taken (as indicated in Section 4.5). Since the mapping of the requirement matrix into a flow for a fixed set of parameters, $\{\lambda_{jk}\}$, is continuous, if R(1) is "close" to R(0), then f(1) is "close" to $f^{*}(0)$, and f(1) is unlikely to be unfeasible. The new minimum delay problem is solved to a 2% tolerance, and the procedure is iterated for R(2), R(3), R(4), and R(5).

Table B.1 summarizes the convergence of the TR to the successive 2% tolerance estimates of the solutions to the given minimum delay problem. The following points are noted. The load on the network is fairly heavy with some channels in each solution having a utilization factor greater than 0.7. Resolving the problem for a perturbation is generally fast compared to solving the problem from an arbitrary starting point depending on the size of the perturbation. For this problem, the degradation of the expected delay from the optimum value would generally be less than 4 or 5% for an optimum routing policy computed 1 second beforehand if $\alpha_t \leq 0.10$ and, at least to the author, this was surprisingly low. It is also noted that the random variables generated for the problem were not exactly Gaussian, and one effect of this was a drift towards a more heavily loaded network

on each successive minimum delay problem.

A parenthetical remark. As one looks at the above problem, naturally the question of the sensitivity of a routing policy to changes in requirements or inaccurate estimates arises. If $f^{\#}$ solves B.1, it is well known that the routing policy which corresponds to $f^{\#}$ is not unique. If one says that he wants the "best" of the many routing policies, then a dilemma arises as to how the best should be defined. On the one hand, the routing policy with the greatest number of "pure" routing strategies has the advantage of reducing out-of-order message deliveries. That is, if the packets belonging to a given message traverse different routes, then their arrival at the destination is generally out of sequence, which complicates the reconstruction of the message. However, it is intuitively clear that the routing policy which is the most split is also the least sensitive to arbitrary perturbations of the requirement matrix. This results in increased intervals between necessary routing updates, which may be interpreted as increasing the stability of the network.

The intuitive clarity of this is enhanced by the following argument. Let each r_{ij} be an independent random variable with $E(r_{ij}) = r$ and $Var(r_{ij}) = \sigma^2$. On the ith channel, perhaps one is given the choice of routing commodity (1, 2) alone, or the sum of the fractions, $1/(N-1)$, of each commodity $(1, k)$ for $k \in \{2, 3, \cdots, N\}$. The expected value of flow on the ith channel is the same in each case, but the variance of the flow is σ^2 for the former and $\sigma^2/(N-1)$ for the latter. It follows that if the same types of choices go on throughout the entire network, and one

always chooses to split routes as much as possible, then one has in some sense minimized the variation of the flow in the network from that which is expected. This, in turn, increases the stability of the network.

	α_t	Value of the expected delay using the set, $\{\lambda_{jk}\}$, of the estimate of the solution to the previous problem.	Value of the expected delay after minimization to a maximal 2% tolerance in the error.	Computation time (seconds) required for the new estimate of the solution of the minimum delay problem using the previous set, $\{\lambda_{jk}\}$, as a starting point for the minimization.
t = 0	-	-	0.63546	0.55
t = 1	0.02	0.63995	0.63949	0.06
t = 2	0.04	0.64743	0.64533	0.09
t = 3	0.06	0.65138	0.64705	0.07
t = 4	0.08	0.66572	0.66373	0.09
t = 5	0.10	0.75788	0.73243	0.17

Table B.1 A summary of the results of the perturbation problem on the seven node symmetric network using the TR algorithm.

APPENDIX C

THE REQUIREMENT MATRICES FOR THE SAMPLE PROBLEMS

This appendix specifies the requirement matrices for the problems of Chapter IV. Some choices made in this appendix are arbitrary. There is an attempt here to explain how they were made and to justify them.

The first consideration was to avoid extraordinary degeneracies that might arise in solutions to 4.1.1. Such degeneracies would arise if one made a typical assumption that all off-diagonal elements of the requirement matrix are equal. Furthermore, the assumption is counter-intuitive when compared to the assumption that all such elements merely have the same statistics. The latter assumption was employed.

In choosing the statistics of the elements, convenience was a primary objective. A computer program devised by Rader [RADE69] which generates uncorrelated, nearly Gaussian pseudo-random variables met this criterion. This allowed regeneration of the matrices for different algorithms that solved the same problems without requiring maintenance of a large table of random variables. Moreover, a simple method exists for perturbing the matrices in a quite arbitrary fashion while retaining the form of the density functions of the demands (see Appendix B).

In keeping with the above, each term r_{ij} of the demand matrix was assumed to be a random variable with the following properties:

(i) r_{ij} and r_{mk} are independent random variables in the case $i \neq m$ or $j \neq k$.

(ii) $r_{ij} = 0$ if $i = j$. Otherwise, the probability density of r_{ij} is given by:

$$p_{r_{ij}}(r_o) = \begin{cases} (2/3\pi)^{\frac{1}{2}} \exp(-r_o^2/6) & \text{for } r_o \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

That is, $r_{ij} = |x_{ij}|$ if $i \neq j$, where x_{ij} is a zero mean Gaussian random variable with a variance of 3. The results of Chapter IV are independent of the value of the variance except as a scaling factor.

Clearly, these properties of the elements of the requirement matrix are not suited to every computer-communication network. Among other possibilities, a probabilistic model of the elements for a particular network might include: (i) dependency among the random variables, (ii) a nonzero probability that $r_{ij} = 0$ of $i \neq j$, (iii) an upper bound on the elements, and (iv) different means, variances, forms of densities, etc., which depend on i and j. Nevertheless, the author is not of the opinion that the algorithms of this thesis are highly dependent on the model of the requirement matrix.

Table C.1 displays a set of nearly independent samples, $\{y_{mk}\}$ (where y_{mk} is the kth element of the mth row of the table), of a random variable y which is zero-mean, approximately Gaussian with a variance of 3 as generated by Rader's algorithm. The requirement matrices of the sample problems in Chapter IV are specified by:

$$r_{ij} = |y_{mk}|$$

such that

$$m = [(j + (i - 1)N)/8]^{+}$$

and

$$k = j + (i - 1)N - 8(m - 1)$$

where $[x]^{+}$ is the smallest integer which is not less than x. The definitions of m and k are simply specifications of the mapping of the table of random variables into the demand matrix.

For example, for the seven node symmetric network, the value r_{35} is desired. Thus,

$$m = [(5 + (3 - 1)7)/8]^{+} = [2.875]^{+} = 3$$

$$k = 19 - 8(3 - 1) = 3$$

which yields $r_{35} = |y_{33}| = 0.11183$.

Table C.1 These values specify the elements of the requirement matrix for the sample problems of Chapter IV according to the rules of Appendix C.

-1.42007	3.95158	-1.79764	-0.90458	-2.40416	-1.39871	1.98901	0.92519	ROW 0001
-0.17576	-3.39102	-2.13337	-0.38331	2.82439	-1.60753	-1.18648	-0.86243	ROW 0002
0.37623	-1.14629	-0.11183	-1.45395	-0.64342	0.68312	0.18868	1.76612	ROW 0003
1.48703	0.48023	-3.20446	2.96163	-3.68120	1.52252	-1.59049	-0.87240	ROW 0004
2.16696	0.70839	0.87840	0.73657	-1.01034	2.25519	0.77410	-1.30088	ROW 0005
0.07169	-2.82485	1.12968	0.43653	-1.18605	1.16184	-0.69941	-1.18680	ROW 0006
0.16772	0.17148	1.45049	1.52352	-4.44804	-0.28550	0.79033	-0.02115	ROW 0007
0.11823	1.73371	3.26226	1.43329	-1.88376	-0.26641	2.20458	3.15341	ROW 0008
-0.50151	0.35109	2.23883	4.37035	-1.30209	-0.46511	0.91997	-1.14838	ROW 0009
-2.65142	-1.07044	-1.53460	0.10473	0.95558	-0.41961	-2.79727	-0.57103	ROW 0010
0.23844	0.00125	0.12787	-2.26067	1.22733	0.16924	1.15573	0.90324	ROW 0011
1.92083	-2.52300	-1.03430	0.89203	1.30688	-1.67180	1.62335	-0.06576	ROW 0012
2.45430	-1.26734	0.37355	2.63551	2.67223	1.02845	-0.71975	-1.59015	ROW 0013
0.83933	1.58380	3.02835	0.31367	-1.65070	0.59510	-1.54541	0.83484	ROW 0014
-1.28344	0.80348	0.25485	0.27393	-0.42886	1.50817	0.63077	-1.83643	ROW 0015
0.70771	-2.32561	4.74949	-2.27374	-3.50262	-1.84863	-2.17005	-2.06773	ROW 0016
3.23069	4.10631	1.04937	-1.01264	-2.15955	0.54373	-2.20283	0.30056	ROW 0017
0.44936	0.40910	-1.21595	3.15953	1.09587	0.99577	-1.22365	-0.27785	ROW 0018
3.05391	-0.47609	-1.97177	-2.07975	1.14707	-0.67570	0.38456	0.11693	ROW 0019
-1.55152	-0.55561	1.37164	-0.58382	0.18368	-3.00370	2.80607	2.02920	ROW 0020
0.16550	-1.26388	0.98143	0.86856	0.23521	-4.04097	-0.59469	0.24241	ROW 0021
-1.42006	-0.47936	1.98352	-2.55158	2.75428	-2.30649	-1.27074	1.07770	ROW 0022
3.14600	0.04364	3.70556	1.11407	2.35657	1.26203	0.61547	-0.84088	ROW 0023
-1.73987	0.85147	-0.73225	-1.67741	-1.47350	-0.95470	2.56987	-1.33845	ROW 0024
-1.65592	1.59382	-1.27154	-1.94128	-2.93046	0.62140	-0.11395	2.60300	ROW 0025
0.30896	-0.55815	-1.02992	-0.28512	0.35643	0.68225	-0.79304	1.37675	ROW 0026
-0.67201	3.45141	1.19074	-4.77153	-0.32352	-1.94886	-3.02031	-2.78624	ROW 0027
-2.04543	1.10656	-2.23805	2.39433	2.89539	0.26095	0.44580	-1.66116	ROW 0028
-1.75467	1.07964	0.61335	-1.53458	0.96386	-0.27694	-0.03791	-0.32529	ROW 0029
-4.05696	-2.34757	-0.69672	-1.60527	-1.59152	-1.88751	2.18839	-1.03383	ROW 0030
-1.65508	1.76587	-3.22695	3.63157	0.43098	-0.06551	-1.42688	-0.77628	ROW 0031
-1.28152	-2.28490	1.44269	1.27482	2.84385	1.37604	0.59594	0.74450	ROW 0032
-0.92913	0.43412	-2.44501	1.32893	2.71196	-2.15915	-1.41017	-2.81964	ROW 0033
0.83488	-2.23589	-1.56159	0.62244	-0.37186	1.54650	1.38074	1.36357	ROW 0034
-2.82406	-0.83471	0.46055	-1.65761	-1.19975	-0.47638	-1.78852	0.40779	ROW 0035
-0.68853	-0.34054	-0.41363	-1.99888	1.50382	-0.28367	-2.44093	1.65931	ROW 0036

-0.98306	1.49135	-0.63902	-3.24590	-0.07579	-0.25025	-0.87550	2.72311	ROW 0037
0.60901	-1.12973	0.37229	-0.72224	0.20054	-0.06016	1.56507	2.03173	ROW 0038
-1.11508	1.48859	-1.20230	0.35468	2.14081	2.84602	-4.75534	0.00280	ROW 0039
1.85851	1.11784	-0.14381	-3.07138	-1.31845	0.21857	0.24032	-0.30748	ROW 0040
-0.97535	-2.42995	0.90683	-2.86959	2.27753	-3.19334	1.59936	-1.38842	ROW 0041
0.21836	1.24246	0.33170	1.16640	-0.02324	1.88206	-0.47015	1.70534	ROW 0042
-0.42412	-0.09806	-2.05028	2.27275	-0.95039	-1.76896	-1.44679	-1.05341	ROW 0043
-1.62808	4.30738	-0.53703	0.61293	0.62391	-0.26412	0.86413	0.12650	ROW 0044
1.98965	3.79018	-2.19357	-0.80035	-0.13471	1.11893	-3.44610	-1.14483	ROW 0045
-1.58423	0.93221	0.23344	-0.67002	-0.80488	-1.88513	-5.96938	-0.15192	ROW 0046
-2.30235	-0.53398	0.07436	-1.76428	-3.42260	3.85440	2.33193	-0.87704	ROW 0047
1.80936	4.64100	0.88135	0.47423	-2.68694	-1.29061	0.86566	1.00161	ROW 0048
-0.65204	1.90199	1.95551	0.10757	-0.57860	-1.46667	-0.39620	1.21101	ROW 0049
-0.19290	-0.61264	3.39583	-0.41268	1.20529	0.46974	1.80754	-0.38742	ROW 0050
-0.16875	1.29665	1.64022	0.01562	-0.34017	0.53797	0.31275	-2.38509	ROW 0051
0.28715	-0.00788	0.34037	-3.33805	0.25418	1.26097	1.18023	-0.12927	ROW 0052
-1.03583	-0.29767	3.52793	0.50894	1.62895	1.29215	-2.66728	3.97273	ROW 0053
1.31245	0.21241	-1.02148	-3.16217	0.88281	4.47088	-0.91499	-0.06028	ROW 0054
4.29952	-2.47389	-0.89112	-4.76858	0.60952	-2.20637	0.17397	-0.36433	ROW 0055
1.13188	3.31025	-0.43201	0.11692	-0.48378	-2.38483	-1.58248	-2.50385	ROW 0056
0.06275	0.28584	0.24584	-0.42719	0.36261	-2.32064	-0.59260	-1.21118	ROW 0057
0.65459	-0.18246	2.10271	-0.04049	0.13237	1.34957	-2.05550	0.05837	ROW 0058
-0.58284	2.47272	-1.94703	-1.11208	-0.31838	-0.29100	-0.73096	-0.22783	ROW 0059
-0.76819	-0.85838	2.27772	4.08124	-1.99394	-1.41732	-0.55137	1.24339	ROW 0060
1.24598	-2.64680	2.02464	-1.35529	1.02047	0.63871	-0.72098	-0.22430	ROW 0061
-2.10792	1.87114	2.95699	-3.34388	-2.87911	1.52177	-0.23632	2.82464	ROW 0062
2.34733	0.78227	-0.84486	0.20186	-0.97309	-0.74110	1.79066	2.96114	ROW 0063
1.50861	0.71896	0.53509	1.24408	2.60683	-0.39416	3.24662	3.58698	ROW 0064
2.59402	-0.90434	-1.01238	-3.34474	0.00005	1.31287	1.17161	-0.39975	ROW 0065
0.49375	3.17262	2.10728	-1.10068	1.58330	0.60953	2.73702	-3.16021	ROW 0066
0.14354	1.55436	-0.22053	0.00093	-2.37777	-2.79068	-1.49001	-1.18936	ROW 0067
1.90924	-0.68366	1.10401	-1.33190	3.86521	0.27101	-1.06306	-1.69631	ROW 0068
-0.53324	4.58368	-0.53619	-0.26897	-1.20933	-2.21899	-1.45671	2.77477	ROW 0069
1.23106	-0.77727	-0.47499	0.70412	0.40474	1.65721	-0.35260	-0.32436	ROW 0070
2.48759	-0.61883	2.93192	2.52483	1.59767	-0.08194	-2.29183	0.89929	ROW 0071
3.12164	-1.12362	3.92444	-1.35848	1.39895	1.05521	-1.16340	3.37485	ROW 0072

1.32967	-2.25245	3.61348	2.14410	-1.65485	2.63035	-0.00806	0.59124	ROW 0073
1.98674	0.45001	0.45804	1.42605	2.07677	1.80767	0.65686	1.73040	ROW 0074
0.08203	2.56518	-1.55936	0.13109	-1.67856	-2.11939	-1.12236	-3.18399	ROW 0075
0.04667	-0.82553	2.41037	2.37184	2.36892	-0.59867	-1.40317	-2.49564	ROW 0076
-0.49166	-2.83084	-0.24942	0.07190	0.31158	2.86855	-0.67771	0.87740	ROW 0077
0.28420	1.66641	-0.78405	0.71916	0.96035	0.96746	-2.00966	-1.28889	ROW 0078
2.47287	-2.79913	-0.08257	1.71619	-0.28356	-0.18584	1.50709	-1.05591	ROW 0079
-0.34824	1.70789	-1.95039	-2.98217	-1.05333	3.14656	-0.49667	-0.50562	ROW 0080
-5.06723	-2.58416	1.26910	0.56799	1.31440	-4.93030	0.47399	0.90198	ROW 0081
0.72922	2.26793	-0.75134	1.46985	-3.44091	-0.09503	4.45050	-0.20025	ROW 0082
-3.18735	-0.89377	0.69925	0.57357	3.30072	-0.69684	-1.00817	-1.01359	ROW 0083
-0.90554	0.61237	1.54797	0.76746	1.33728	-3.07216	-2.09797	0.18190	ROW 0084
2.44513	2.36178	1.81368	0.37505	-0.76378	1.20404	-0.77536	1.83898	ROW 0085
-0.21358	1.90896	2.85395	2.71084	0.31836	-0.90369	0.69852	-1.07252	ROW 0086
-1.42723	-1.32154	-1.20718	0.82192	0.37063	0.01509	-1.35157	-2.82835	ROW 0087
2.17081	-1.45193	0.75118	-0.64209	-0.60029	-0.40848	0.01633	-0.34387	ROW 0088
-1.24439	-1.04497	-1.89568	-0.58213	-1.84456	1.04097	1.59912	-1.90599	ROW 0089
1.33455	-1.07629	1.24326	-0.69840	-1.55171	-1.43869	2.77345	-0.43151	ROW 0090
1.01262	-3.25799	0.99120	4.04213	-0.28167	-1.29426	2.16569	0.01857	ROW 0091
-1.53904	-3.73014	-2.75881	-2.29784	2.28157	-0.49844	-1.99926	-3.44533	ROW 0092
2.53615	-0.42406	1.73562	0.12938	0.85253	-3.52684	0.61152	0.16765	ROW 0093
-0.98212	1.46833	-0.75499	1.35796	-0.49081	-3.34602	-0.25126	0.40245	ROW 0094
0.37403	2.31925	-1.60700	-0.63877	0.29489	1.33718	-0.67013	-0.28613	ROW 0095
1.51974	-0.41363	-7.46369	0.99621	1.62209	-1.42983	0.21491	0.92507	ROW 0096

APPENDIX D

DETERMINATION OF ROUTING FOR HIGHER ORDER CRITERIA

This appendix presents one example of a linear routing problem solved for higher order criteria. The results which are presented are typical of those problems for which higher order criteria were considered. The example is of the ten node problem considered in Section 4.3 and redrawn in Fig. D.1. In that figure, along each arrowed arc is a pair (i, β) where i is the first criterion for which the respective channel was associated with a negative dual variable, and β is the value of $\hat{\beta}_1$. The problem was solved through the fourth criterion.

Table D.2 presents the computation times for the various criteria with the LTR algorithm. Each iteration of the linear programming results which is listed represents one of two types of events:

(i) If the iteration is within the same criterion level as the previous iteration, columns have been added and the linear program has been reminimized. For example, on the 5th iteration ten columns (which in this case are trees) were added to the program, and the best value of the objective, subject to the 31 columns presently available, was determined as 10.48.

(ii) If the iteration is on a different criterion level

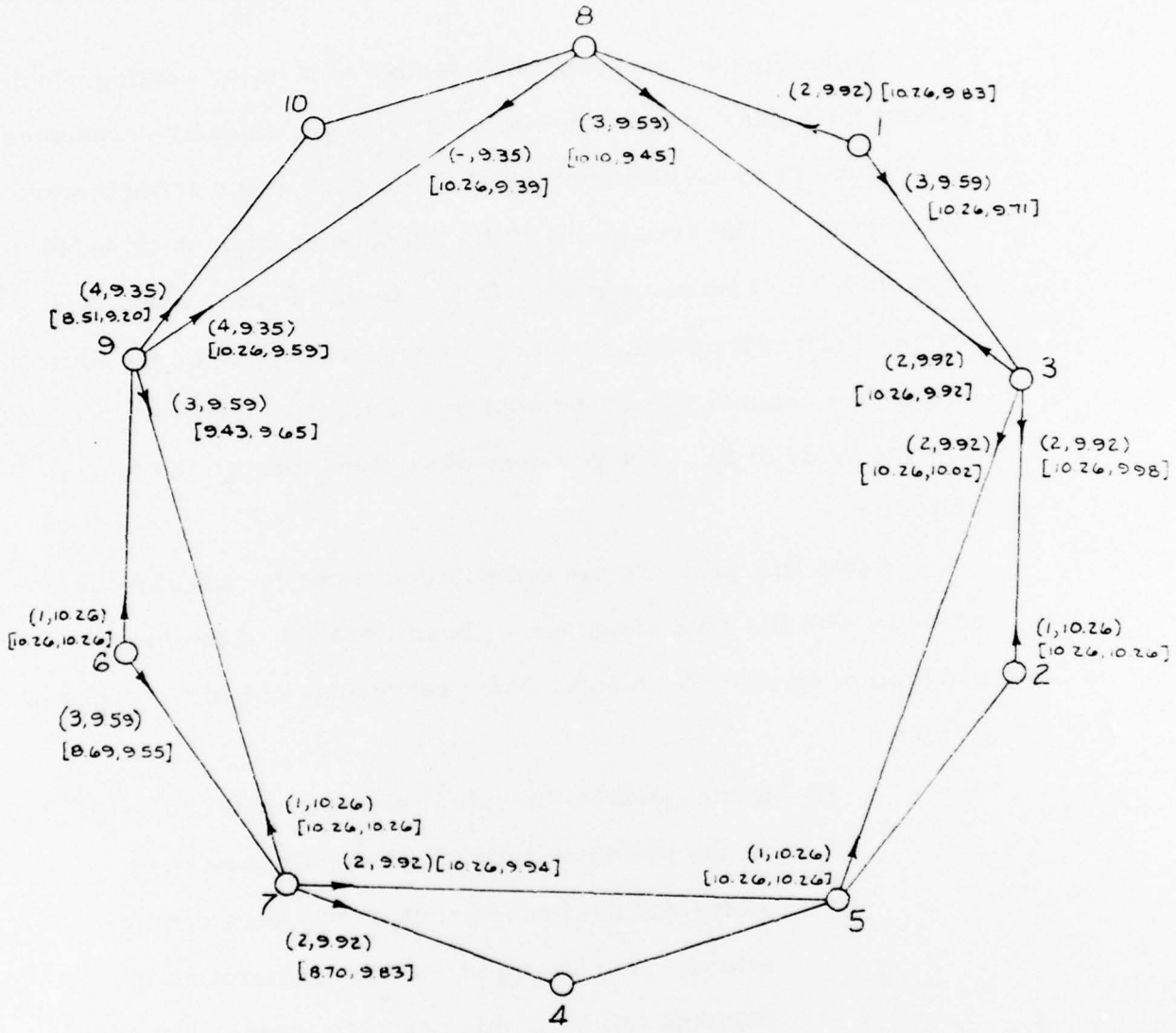


Fig. D.1 A ten node network with thirty directed arcs.

Criterion, i	Iteration	Value of the objective, β_i	Cumulative computation time (seconds)	Number of columns in the program
1	1	15.16	0.01	10
	2	14.97	0.03	14
	3	14.06	0.05	18
	4	12.30	0.12	21
	5	10.48	0.29	31
	6	10.26	0.42	41
2	7	10.095	1.25	41
	8	10.091	1.34	46
	9	9.96	2.02	56
	10	9.92	2.21	66
3	11	9.606	3.47	66
	12	9.605	3.53	69
	13	9.59	4.52	72
4	14	9.35	7.24	72

Table D.1 Computation time for the LTR algorithm to solve the linear routing problem associated with Fig. D.1 through the fourth criterion.

from the previous iteration, the program was changed to reflect the new criterion, and the new objective minimized with the columns existing from the previous iteration.

It is noted that the second type of iteration is particularly time consuming. This reflects the large number of basis changes required to accomplish the minimization. Moreover, as the number of the criterion increases, the number of active constraints (which is directly related to the size of the basis inverse) generally increases as well. This is because every channel which had its utilization factor fixed by a previous criterion corresponds to a necessarily tight constraint for the current program. The consequence is that each successive criterion generally requires more computation time than its predecessor.

Also in Fig. D.1, there corresponds to many of the channels a pair [x, y] where:

x = the value of the flow in the channel at the conclusion of the 6th iteration of the LTR algorithm.

y = the value of the flow in the channel at the conclusion of the TR* algorithm (i.e. after 1 second of computation time) which produced the appropriate curve in Fig. 4.5.1.

The pairs are provided for those channels for which either x or y is greater than 9. Observe the following: for each set $\{(i, \hat{\beta}_i), [x, y]\}$ along each channel in Fig. D.1, $|y - \hat{\beta}_i| \leq |x - \hat{\beta}_i|$. This verifies the

suggestion at the conclusion of Section 4.5 that the $AL^*(\alpha, \epsilon, \gamma)$ algorithm tends to produce a solution that is better from the point of view of higher order criteria than do linear programs which are solved only for the first criterion.

APPENDIX E

FURTHER COMMENTS ON THE LOWER BOUND FOR THE LINEAR ROUTING PROBLEM

It is noted in the discussion of Fig. 4.5.2 that the respective lower bounds for the TR*, EF*, and CH* algorithms are not as satisfactory as those of the other examples. This is true even though the lower bound can become exact. The underlying reasons for this phenomenon are discussed in Sections 4.4 and 4.5. This appendix elaborates on that discussion and demonstrates the possibility of algorithms that do somewhat better.

The TR* algorithm was run for 1.08 seconds on the linear routing problem associated with Fig. 4.3.2, at which point β_1 was 91.73, and the values of the eleven most utilized channels were as follows:

Channel	Utilization, f_i/C_i
(20, 25)	91.73
(24, 4)	91.58
(14, 15)	87.43
(25, 20	86.91
(15, 16)	86.73
(20, 22)	86.15
(22, 20)	86.09
(23, 13)	85.33
(1, 2)	83.80
(4, 24)	83.05
(15, 14)	82.61

To look at this list and conclude, simply from the values of utilization,

that the top three channels are the bottleneck would be absurd. However, if the reader refers to Fig. E.1 and indicates the above channels on the graph in the order of appearance in the table, the following observations are clear:

(i) The first cutset occurs after the third channel is considered. One would therefore say that this is the first estimate of the bottleneck with any potential for producing a good lower bound on β_1.

(ii) The addition of channel (25, 20) by itself will degrade the bound. This is because there would also be a path with zero estimated bottleneck channels between nodes 25 and 20, i.e. the addition of channel (25, 20) could not form or contribute to the bottleneck.

(iii) The addition of channel (15, 16) to the first three (or the first four) channels will not improve $\beta_L(S, \infty, f)$. Two channels in a simple sequence like (14, 15) and (15, 16) are not beneficial for the reasons discussed in Section 4.4

The list of observations can go on; however, it seems to become rapidly evident that the three most utilized channels are most probably the bottleneck, and the calculation for the lower bound based on any other estimate of the bottleneck is probably in vain.

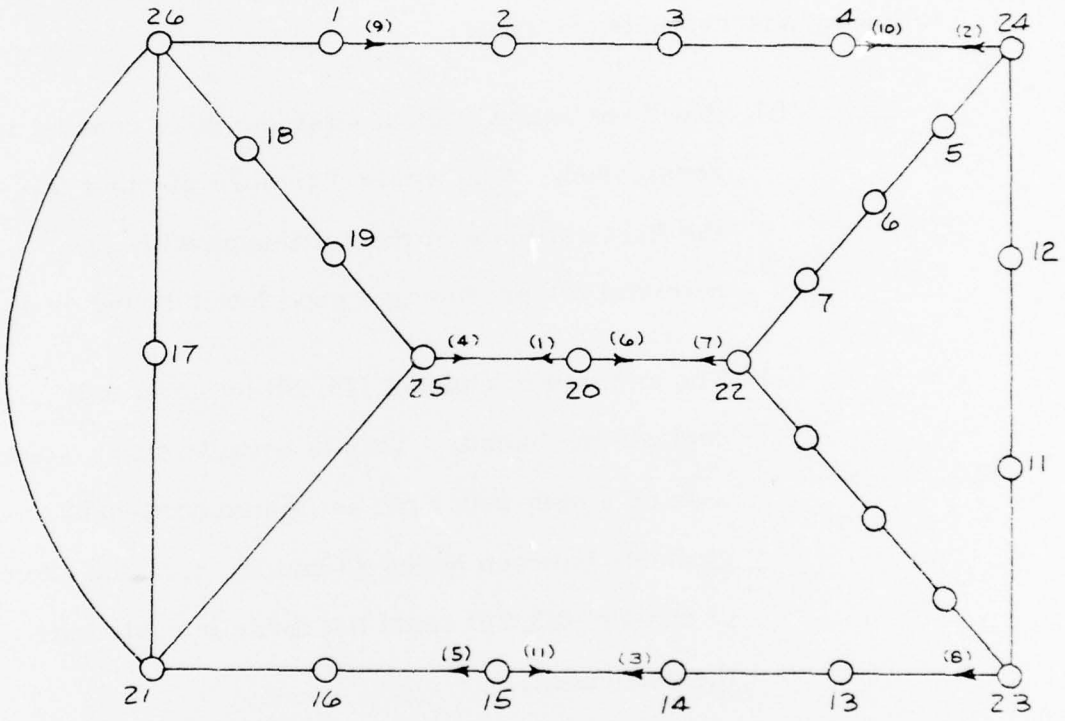


Fig. E.1 The 26 node network of Fig. 4.3.2. Along each arrowed channel there is a number (k) where k is the order of the appearance of the channel in the utilization table of Appendix E.

To put this kind of subjective a posteriori intelligence into a computer program which estimates the bottleneck would be of great benefit, if its execution required little computation time compared to a shortest route computation. The method which the author instructed the TR* algorithm to use for choosing S, for the previously described run, was not very sophisticated. It was as follows: (i) choose S to be all channels with a utilization factor greater than $(0.9)\beta_1$, (ii) calculate a lower bound, (iii) reduce the cardinality of S by a factor of 0.75 (rounding to the next lower integer if required) by rejecting the least utilized channels, and (iv) if less than 5 lower bounds have been calculated, then return to the second step. The calculated lower bounds were:

$\|S\|$	$B_L(S, \infty, f)$
11	57.86
8	44.23
6	49.98
4	66.84
3	89.24

The last of these values is in fact $\hat{\beta}_1$.

Although this appendix probably asks more questions than it answers, it is meant to suggest that there is a great deal of information on the identity of the bottleneck when the $AL^*(\alpha, \epsilon, \gamma)$ algorithm reaches the knee of its convergence. Much more information is available than is used to produce the lower bounds in Figs. 4.5.1, 4.5.2, 4.5.3, and 4.5.4. The problem, of course, is how to extract that information in an efficient manner.

BIBLIOGRAPHY

[ASSA75] A. Assad, "Multicommodity network flows--a survey", Massachusetts Institute of Technology, Operations Research Center, OR 045-75, December 1975.

[BASK75] F. Baskett, K. Chandy, R. Muntz, and F. Palacios, "Open, closed, and mixed networks of queues with different classes of customers", J. Assoc. Comp. Mach., Vol. 22, No. 2, pp. 248-60, 1975.

[BEST75] M. Best, "Optimization of nonlinear performance criteria subject to flow constraints", Proc. 18th Midwest Symp. on Circuit Theory, 1975.

[BURK56] P. Burke, "The output of a queueing system", Operations Research, pp. 699-704, 1956.

[CANT74] D. Cantor and M. Gerla, "Optimal routing in a packet-switched computer network", IEEE Trans. Computers, Vol. C-23, No. 10, October 1974.

[DAFE71] S. Dafermos, "An extended traffic assignment model with applications to two-way traffic", Trans. Sci., Vol. 5, pp. 366-89, 1971.

[DANT63] G. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, N. J., 1963.

[DANT66] G. Dantzig, All Shortest Routes in a Graph, Operations Research House, Stanford University, Technical Report 66-3, November 1966.

[DIJK59] E. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, Vol. 1, pp. 269-271, 1959.

[DREY69] S. Dreyfus, "An appraisal of some shortest-path algorithms", Opns. Res., Vol. 17, pp. 395-411, 1969.

[FLOY62] R. Floyd, "Algorithm 97, shortest path", Comm. ACM, Vol. 5, pp. 345, 1962.

[FRAN71a] H. Frank and W. Chou, "Routing in computer networks", Networks, Vol. 1, pp. 99-122, 1971.

[FRAN71b] H. Frank and I. Frisch, Communication, Transmission, and Transportation Networks, Addison-Wesley, Reading, Mass., 1971.

[FRAT73] L. Fratta, M. Gerla and L. Kleinrock, "The flow deviation method: an approach to store-and-forward communication network design", Networks, Vol. 3, No.2, pp. 97-133, 1973.

[FULT72] G. L. Fultz, "Adaptive routing techniques for message switching computer-communication networks", School Eng. and Appl. Sci., U.C.L.A., UCLA-ENG-7252, July 1972.

[GALL77] R. Gallager, "A minimum delay routing algorithm using distributed computation", IEEE Trans. Comm., Vol. COM-25, No.1, January 1977.

[GOLD75] B. Golden, "A minimum cost multicommodity network flow problem concerning imports and exports", Opn. Res. Center, MIT, Tech. Report 105, September 1975.

[GOLD76] B. Golden, "Shortest path algorithms: a comparison", Operations Research Center Technical Note, MIT, 1976.

[HADL64] G. Hadley, Non-Linear and Dynamic Programming, Addison-Wesley, Reading, Mass., 1964.

[HOFF76] A. Hoffman and S. Winograd, "Finding all shortest distances in a directed network", IBM J. Res. Develop., Vol. 16, pp. 412-414, July 1972.

[HU69] T. C. Hu, Integer Programming and Network Flows, Addison-Wesley, Reading, Mass., 1969.

[HU71] T. C. Hu, "A decomposition algorithm for shortest paths in a network", Opns. Res., Vol. 16, No.1, pp. 91-102, 1971.

[JACK57] J. Jackson, "Networks of Waiting Lines", Operations Research, Vol. 5, pp. 518-21, 1957.

[JACK63] J. Jackson, "Jobshop-like queueing systems", Management Science, Vol. 10, No.1, pp. 131-42, October 1963.

[KLEI64] L. Kleinrock, Communication Nets: Stochastic Message Flow and Delay, McGraw-Hill, 1964.

[KLEI70] L. Kleinrock, "Analytic and simulation methods in computer network design", Conf. Rec., Spring Joint Comput. Conf., AFIPS Conf. Proc., pp. 569-79, 1970.

[KLEI73] L. Kleinrock, "Scheduling, Queueing, and Delays in Time-Shared Systems and Computer Networks", Computer-Communication Networks, Abramson, N. and Kuo, F., Eds., Prentice-Hall, 1973.

[KLEI76] L. Kleinrock, Queueing Systems: Volume 2: Computer Applications, John Wiley & Sons, 1976.

[KLES74] R. W. Klessig, "An algorithm for nonlinear multi-commodity flow problems", Networks, Vol. 4, pp. 343-355, 1974.

[KOBA77] H. Kobayashi and A. Konheim, "Queueing models for computer communication system analysis", IEEE Trans. Communications, Vol. COM-25, No. 1, January 1977.

[LEBL74] L. LeBlanc, E. Morlok, W. Pierskalla, "An accurate and efficient approach to equilibrium traffic assignment on congested networks", Trans. Res. Record 491, Interactive Graphics and Trans. Systems Planning, 12-33, 1974.

[MARS74] R. Marsten, "Users manual for SEXOP", Release 4, Sloan School of Management, MIT, February 1974.

[MCGR74] P. McGregor, "Load Sharing in a Computer Network", Ph.D. Thesis, Dept. of E.E., Polytechnic Institute of New York, June 1974.

[MCGR75] P. McGregor and R. Boorstyn, "Load Sharing in a Computer Network", Int. Conf. on Comm., San Francisco, June 16-18, pp. 41-14 to 41-19, 1975.

[MUNT73] R. Muntz, "Poisson departure processes and queueing networks", Proc. 7th Annual Princeton Conf. Info. Sci. Syst., March 1973.

[NAC71] NAC 4th Semiannual Tech. Report Project "Analysis and optimization of store-and-forward computer networks", Def. Doc. Center, Alexandria, Va., December 1971.

[NGUY74] S. Nguyen, "A unified approach to equilibrium methods for traffic assignments", Int. Symp. on Traffic Equilibrium Methods, Montreal, Quebec, November 1974.

[PIER75] A. R. Pierce, "Bibliography on algorithms for shortest path, shortest spanning tree, and related circuit routing problems", Networks, Vol. 5, No. 2, April 1975.

[RADE69] C. Rader, "A new method of generating Gaussian random variables by computer", MIT Lincoln Laboratory, Tech. Note 1969-49, 1969.

[SCHW76] M. Schwartz and C. Cheung, "The gradient projection algorithm for multiple routing in message-switched networks", IEEE Trans. on Communications, Vol. COM-24, No. 4, April 1976.

[SPIR73] P. Spira, "A new algorithm for finding shortest paths in a graph of positive arcs in average time $o(n^2 \log^2 n)$", SIAM J. Comput., Vol. 2, No. 1, March 1973.

[VARA72] P. Varaiya, Notes on Optimization, Van Nostrand Reinhold, New York, N.Y., 1972.

[WUND75] E. Wunderlich, Load Sharing in a Computer-Communication Network, S.M. Thesis, MIT, Cambridge, Massachusetts, September 1975.

[WUND76] E. Wunderlich, "Load Sharing in a Computer-Communication Network", MIT, Electronic Systems Laboratory Technical Report ESL-R-678, Cambridge, Mass., August 1976.

[YEN71] J. Yen, "On Hu's decomposition algorithm for shortest paths in a network", Opns. Res., Vol. 19, No. 3, pp. 983-985, 1971.

[YEN72] J. Yen, "Finding the lengths of all shortest paths in n-node non-negative-distance complete networks using $\frac{1}{2}n^3$ additions and n^3 comparisons", J. Assoc. Comput. Mach., Vol. 19, No. 3, pp. 423-424, July 1972.

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Assistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Office of Naval Research Code 1021P Arlington, Virginia 22217	6 copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, California 91106	1 copy
New York Area Office (ONR) 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 copies

Dr. A. L. Slafkosky — 1 copy
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380

Office of Naval Research — 1 copy
Code 455
Arlington, Virginia 22217

Office of Naval Research — 1 copy
Code 458
Arlington, Virginia 22217

Naval Electronics Laboratory Center — 1 copy
Advanced Software Technology Division
Code 5200
San Diego, California 92152

Mr. E. H. Gleissner — 1 copy
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084

Captain Grace M. Hopper — 1 copy
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D.C. 20350

Mr. Kin B. Thompson — 1 copy
Technical Director
Information Systems Division (OP-91T)
Office of Chief of Naval Operations
Washington, D.C. 20350

Advanced Research Projects Agency — 1 copy
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209

ORSA Officer (DRSEL-SA) — 1 copy
U.S. Army Electronics Command
Fort Monmouth, N.J. 07703